Programming with Invariants through Targeted Synthesis

John Sarracino

Programming is Hard

Programming is Hard

Therac 25 — 1982

Programming is Hard

Ariane 5 —1996

Programming is Hard

MAERSK
LINE

We are sorry but maerskline.com is temporarily unavailable

We confirm that some Maersk IT systems are down. We are assessing the situation. The safety of your business and our people is our top priority. We will update when we have more information.

We apologize for any inconvenience this causes you.

Maersk Line team

Maersk — 2017

Programming is Hard

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE **SECURITY** DEVOPS BUSINESS PERSONAL TECH SCIENCE EM

Security

It's March 2018, and your Windows PC can be pwned by a web article (well, none of OURS)

Windows — 2018

Programming is Hard

The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE **SECURITY** DEVOPS BUSINESS PERSONAL TECH SCIENCE EM

Security

It's March 2018, and your Windows PC can be pwned by a web article (well, none of OURS)

Windows — always

Computers are a Sadness
and I am the cure

Programs are a Sadness
and I am the cure

Software Invariants

```
ubi_assert(e->num != e1->num);
```

```
if (keyProto.getKeyValue().size() !=
    Ed25519Verify.PUBLIC_KEY_LEN) {
   throw new
   GeneralSecurityException("invalid Ed25519
      public key: incorrect key length");
}
```

```
ensure
   cat_set : has_category(cat)
end
```

Software Invariants

C

```
ubi_assert(e->num != e1->num);
```

```
if (keyProto.getKeyValue().size() !=
    Ed25519Verify.PUBLIC_KEY_LEN) {
   throw new
   GeneralSecurityException("invalid Ed25519
      public key: incorrect key length");
}
```

```
ensure
   cat_set : has_category(cat)
end
```

Software Invariants

```
ubi_assert(e->num != e1->num);
```

C

```
if (keyProto.getKeyValue().size() !=
    Ed25519Verify.PUBLIC_KEY_LEN) {
   throw new
   GeneralSecurityException("invalid Ed25519
      public key: incorrect key length");
}
```

Java

```
ensure
   cat_set : has_category(cat)
end
```

Software Invariants

```c
ubi_assert(e->num != e1->num);
```

C

```java
if (keyProto.getKeyValue().size() !=
    Ed25519Verify.PUBLIC_KEY_LEN) {
   throw new
   GeneralSecurityException("invalid Ed25519
      public key: incorrect key length");
}
```

Java

```
ensure
   cat_set : has_category(cat)
end
```

Eiffel

Software Invariants

```
ubi_assert(e->num != e1->num);
```

C

```
if (k
    E
  th
  Ge

}
```

a

Language agnostic,
Manually maintained.

```
ensure
   cat_set : has_category(cat)
end
```

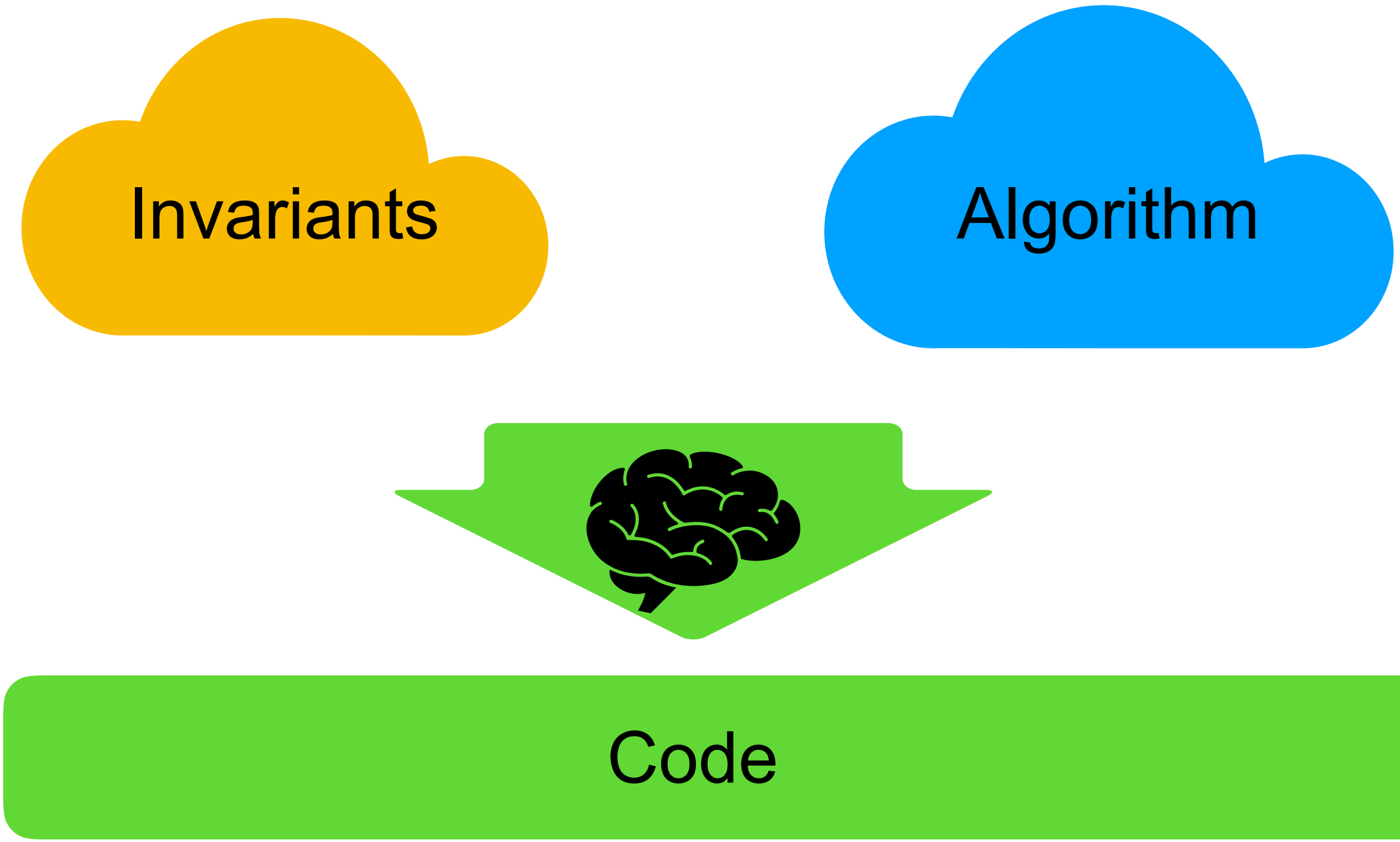
Eiffel

Manual Invariant Programming

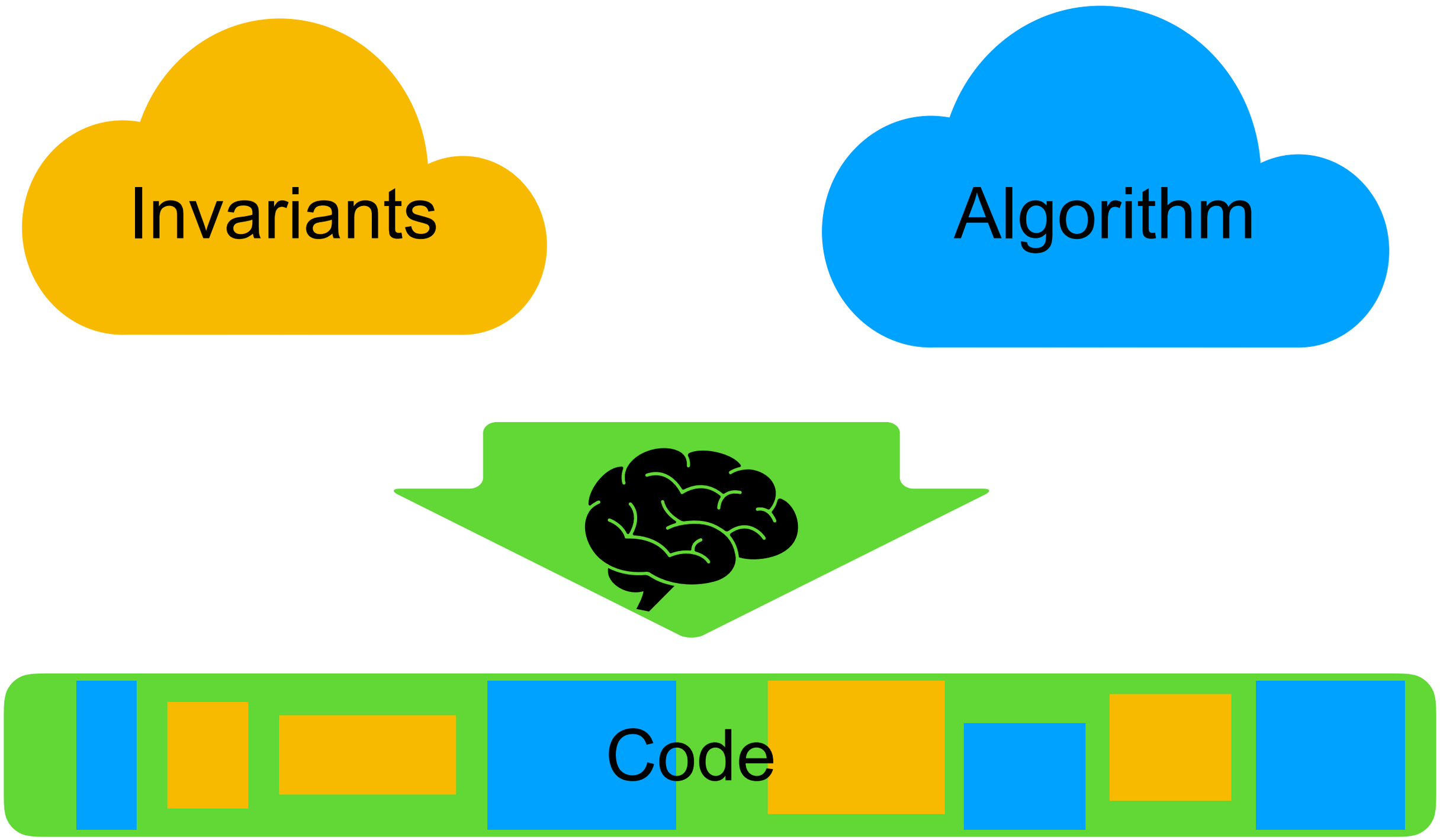
Invariants

Algorithm

Manual Invariant Programming



Manual Invariant Programming



Why are Manual Invariants difficult?

Code changes =>
Programmer checks, maintains invariant

Invariant changes =>
Programmer checks, maintains code

The code change problem: Maintaining Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:



        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant: “All urls are in the urlset”

The code change problem: Maintaining Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant: “All urls are in the urlset”

The code change problem: Maintaining Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']

        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant:

"All urls are in the urlset"

The code change problem: Maintaining Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
```

Tedious and error-prone, particularly with a large codebase

Invariant: "All urls are in the urlset"

The invariant change problem: Maintaining Code

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant:

"All urls are in the urlset"

The invariant change problem: Maintaining Code

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant:

"All urls are in the urlset"

"The allocset contains the urlset "

The invariant change problem: Maintaining Code

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
```

Even more tedious and error-prone

Invariant:

"All urls are in the urlset"

"The allocset contains the urlset "

My thesis statement

Compiling and generating invariants is useful, novel, and broadly applicable.

My thesis proposal

Compiling and generating invariants is useful, novel, and broadly applicable.

I propose a general framework for maintaining invariants

Proposed Work

- Data invariant calculus
- Extensions
- Invariants for Security
- GUI invariants

Outline

- Data invariant calculus
- Extensions
- Invariants for Security
- GUI invariants

Outline

- **Data invariant calculus**
- Extensions
- Invariants for Security
- GUI invariants

Budgeting Example

Weekly	Monthly	Yearly
-200	-800	-9600
-50	-200	-2400
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$

$$\forall i. 12 * months[i] = years[i]$$

Budgeting Example

Weekly	Monthly	Yearly
-200	-800	-9600
-50	-200	-2400
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$

$$\forall i. 12 * months[i] = years[i]$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
-50	-200	-2400
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$

$$\forall i. 12 * months[i] = years[i]$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
-50	-200	-2400
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
-50	-200	-2400
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$

$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$

$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Budgeting Code

```
for (let index = 0; index != week.length; ++index) {

  week[index].oninput = () => {
    let w = parseFloat(week[index].textContent);
    month[index].textContent = w * 4;
    year[index].textContent = w * 12 * 4;
    refreshColors(index);
  }

  month[index].oninput = () => {
    let m = parseFloat(month[index].textContent);
    week[index].textContent = m / 4;
    year[index].textContent = m * 12;
    refreshColors(index);
  }

  year[index].oninput = () => {
    let y = parseFloat(year[index].textContent);
    week[index].textContent = y / (4 * 12);
    month[index].textContent = y / 12;
    refreshColors(index);
  }
}
```

```
document.getElementById('evenPicker').oninput = () => {
  evenColor = document.getElementById('evenPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
}

document.getElementById('oddPicker').oninput = () => {
  oddColor = document.getElementById('oddPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
```

Budgeting Code

```
for (let index = 0; index != week.length; ++index) {

  week[index].oninput = () => {
    let y = parseFloat(week[index].textContent);
    mon
    yea
    ret
  }

  month
    let
    wee
    yea
    ret
  }

  year[index].oninput = () => {
    let y = parseFloat(year[index].textContent);
    week[index].textContent = y / (4 * 12);
    month[index].textContent = y / 12;
    refreshColors(index);
  }
}
```

```
document.getElementById('evenPicker').oninput = () => {
  evenColor = document.getElementById('evenPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
```

Tedious even for this simple example

```
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
```

Budgeting Code

Invariants

```
for (let index = 0; index != week.length; ++index) {

  week[index].oninput = () => {
    let w = parseFloat(week[index].textContent);
    month[index].textContent = w * 4;
    year[index].textContent = w * 12 * 4;
    refreshColors(index);
  }

  month[index].oninput = () => {
    let m = parseFloat(month[index].textContent);
    week[index].textContent = m / 4;
    year[index].textContent = m * 12;
    refreshColors(index);
  }

  year[index].oninput = () => {
    let y = parseFloat(year[index].textContent);
    week[index].textContent = y / (4 * 12);
    month[index].textContent = y / 12;
    refreshColors(index);
  }
}
```

Algorithm

```
document.getElementById('evenPicker').oninput = () => {
  evenColor = document.getElementById('evenPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
}

document.getElementById('oddPicker').oninput = () => {
  oddColor = document.getElementById('oddPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
```

Budgeting Code

Invariants

```
for (let index = 0; index != week.length; ++index) {

  week[index].oninput = () => {
    let w = parseFloat(week[index].textContent);
    month[index].textContent = w * 4;
    year[index].textContent = w * 12 * 4;
    refreshColors(index);
  }

  month[index].oninput = () => {
    let m = parseFloat(month[index].textContent);
    week[index].textContent = m / 4;
    year[index].textContent = m * 12;
    refreshColors(index);
  }

  year[index].oninput = () => {
    let y = parseFloat(year[index].textContent);
    week[index].textContent = y / (4 * 12);
    month[index].textContent = y / 12;
    refreshColors(index);
  }
}
```

Algorithm

```
document.getElementById('evenPicker').oninput = () => {
  evenColor = document.getElementById('evenPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
}

document.getElementById('oddPicker').oninput = () => {
  oddColor = document.getElementById('oddPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
```

Budgeting Code

Invariants

Algorithm

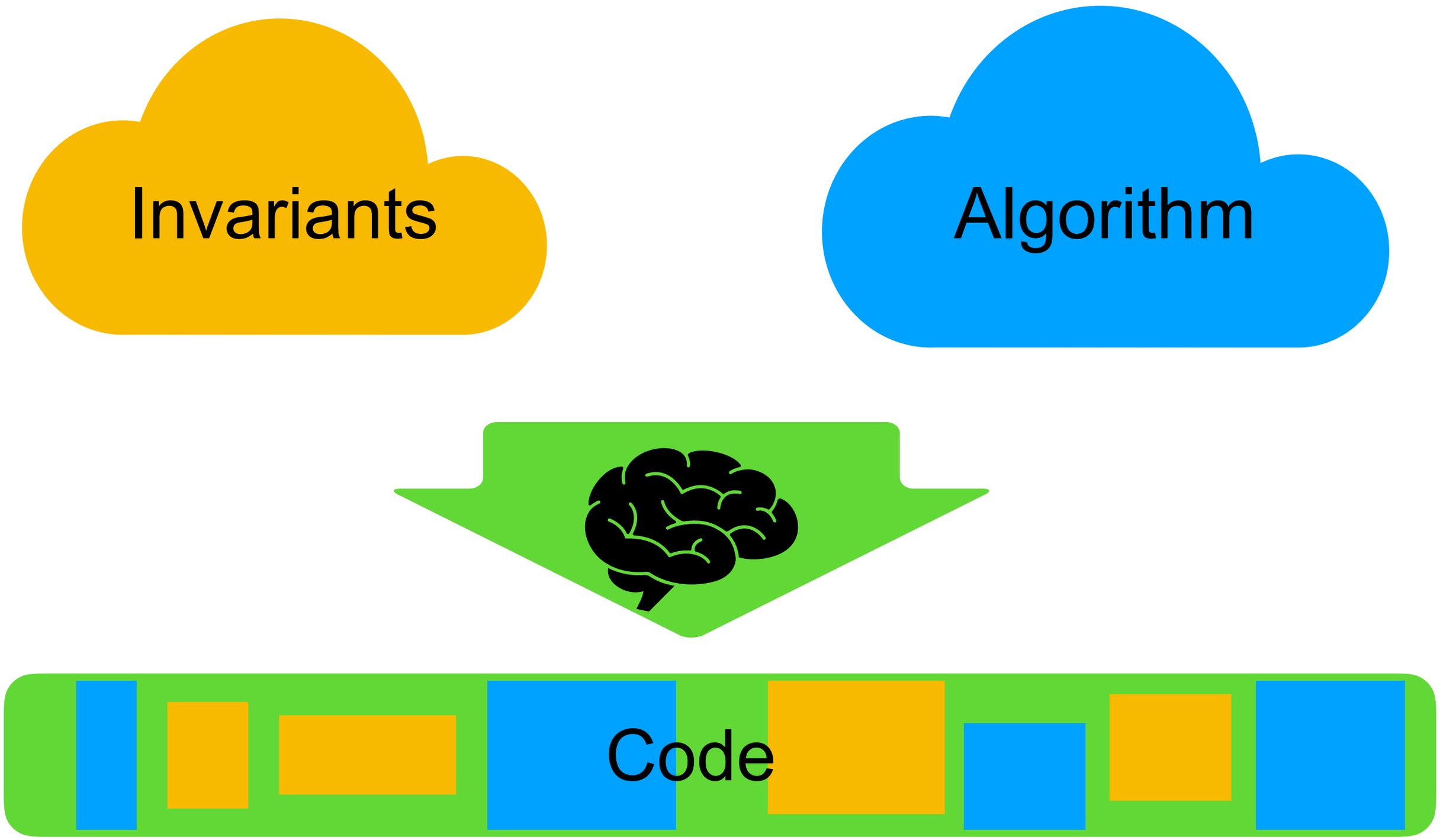
```
for (let index = 0; index != week.length; ++index) {

  week[index].oninput = () => {
    let w = parseFloat(week[index].textContent);
    month[ind
    year[inde
    refreshCo
  }

  month[index
    let m = p
    week[inde
    year[inde
    refreshColors(index);
  }

  year[index].oninput = () => {
    let y = parseFloat(year[index].textContent);
    week[index].textContent = y / (4 * 12);
    month[index].textContent = y / 12;
    refreshColors(index);
  }
}
```

```
document.getElementById('evenPicker').oninput = () => {
  evenColor = document.getElementById('evenPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
```

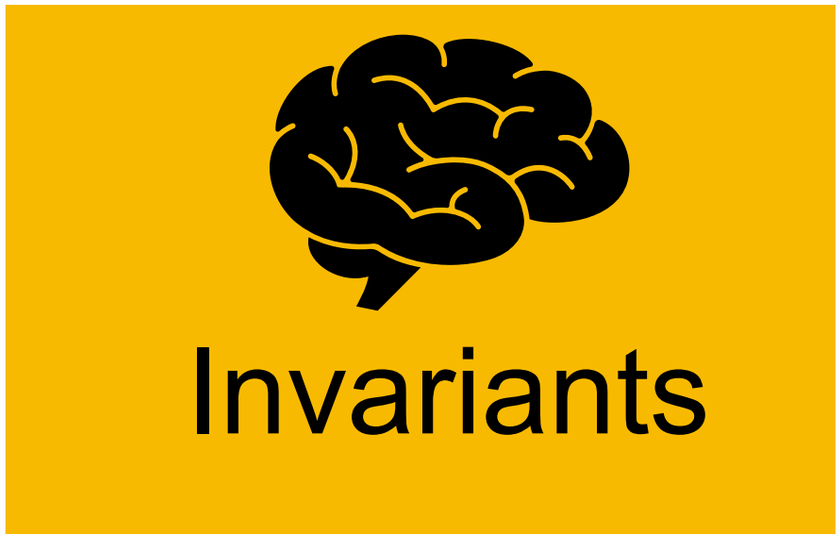
Vast majority of code manually maintains invariant

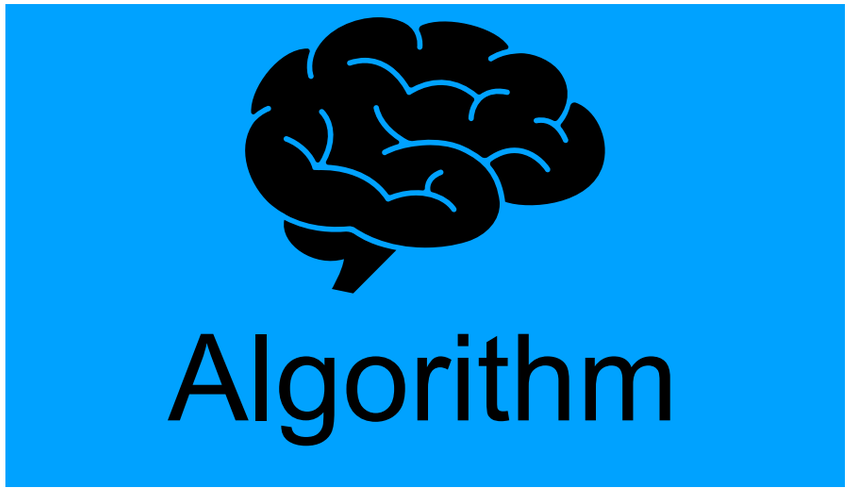
```
document.getElementById('oddPicker').oninput = () => {
  oddColor = document.getElementById('oddPicker').value;
  for (let index = 0; index != week.length; ++index) {
    if (index % 2 == 0) {
      week[index].style.backgroundColor = evenColor;
      month[index].style.backgroundColor = evenColor;
      year[index].style.backgroundColor = evenColor;
    } else {
      week[index].style.backgroundColor = oddColor;
      month[index].style.backgroundColor = oddColor;
      year[index].style.backgroundColor = oddColor;
    }
  }
```

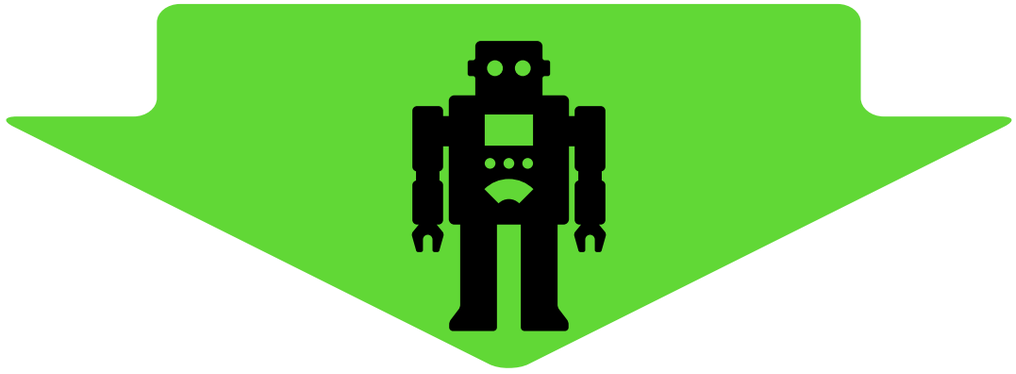
Manual Invariant Programming

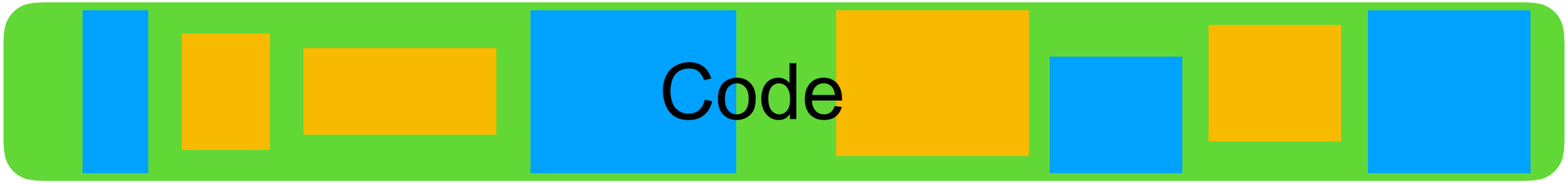


Invariant Synthesis









Language Design Goals

Programmer thinks about invariants once

Compiler automatically handles invariants

Compiling Invariants

Program Synthesis compiles invariants*

Inputs:

1) program with holes e.g. x := ??(y, x)

2) precondition e.g. y = 15

3) requirement e.g. x = y

Compiling Invariants

Program Synthesis compiles invariants*

Inputs:

“expression using x and y” x := ??(y, x)

2) precondition e.g. y = 15

3) requirement e.g. x = y

Language Design Goals

Programmer thinks about invariants once

Compiler automatically handles invariants

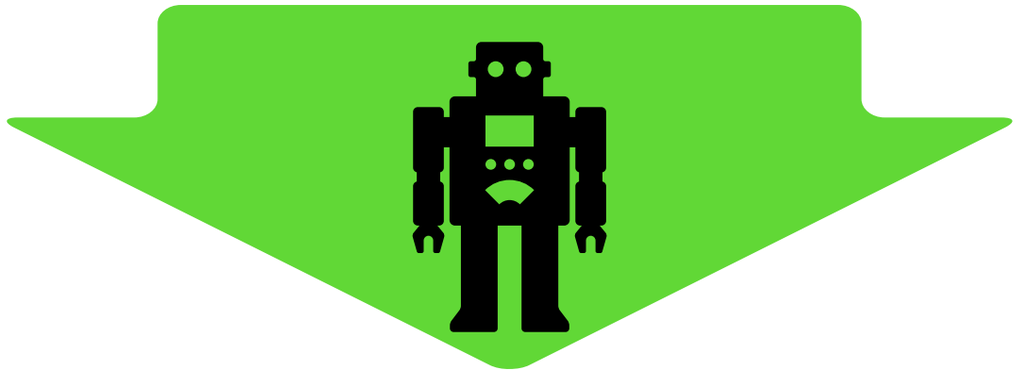
Naive Algorithm

Synthesize at the end of every function.

Use invariants as preconditions, requirements.

Naive Algorithm

```
function updateWeeks(value, i) {
  weeks[i] = value;
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

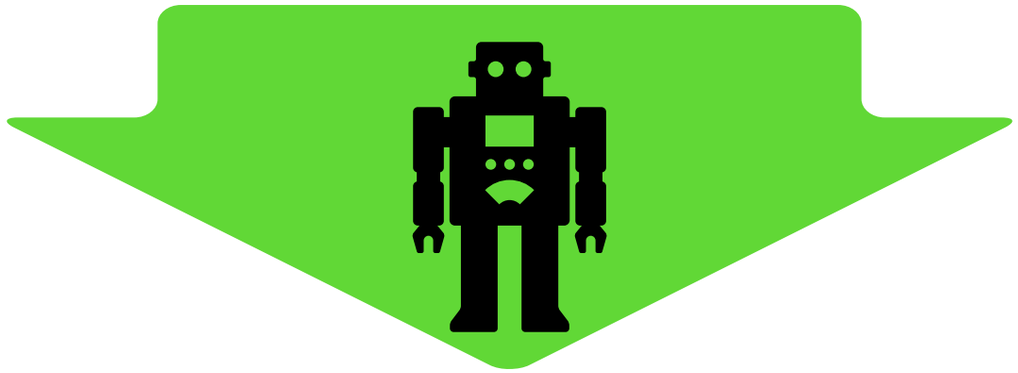
```
weeks[i] = value;
months[i] = ??(value, i, weeks[i]);
years[i] = ??(value, i, weeks[i]);
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Naive Algorithm

```
function updateWeeks(value, i) {
  weeks[i] = value;
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

```
weeks[i] = value;
months[i] = 4*weeks[i];
years[i] = 4*12*weeks[i];
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Naive Algorithm

```
function updateWeeks(value, i) {
  weeks[i] = value;
}
```

Does not scale to realistic programs

Pre

Req

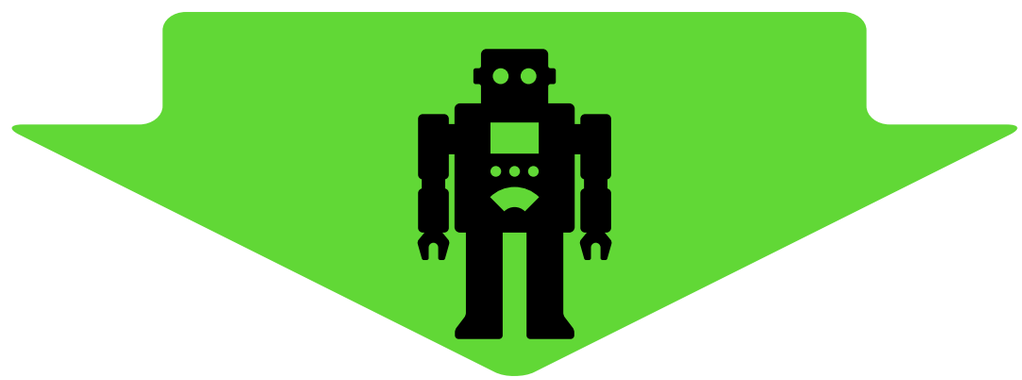
$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

```
weeks[i] = value;
months[i] = ??(value, i, weeks[i]);
years[i] = ??(value, i, weeks[i]);
```

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Problem: Loops don’t scale

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
}
for (let i = 0; i < weeks.length; ++i) {
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Problem: Loops don't scale

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
```

Timeout even for this simple example

Pre

$\forall i. 4 * weeks[i] = months[i]$

$\forall i. 12 * months[i] = years[i]$

```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
}
for (let i = 0; i < weeks.length; ++i) {
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

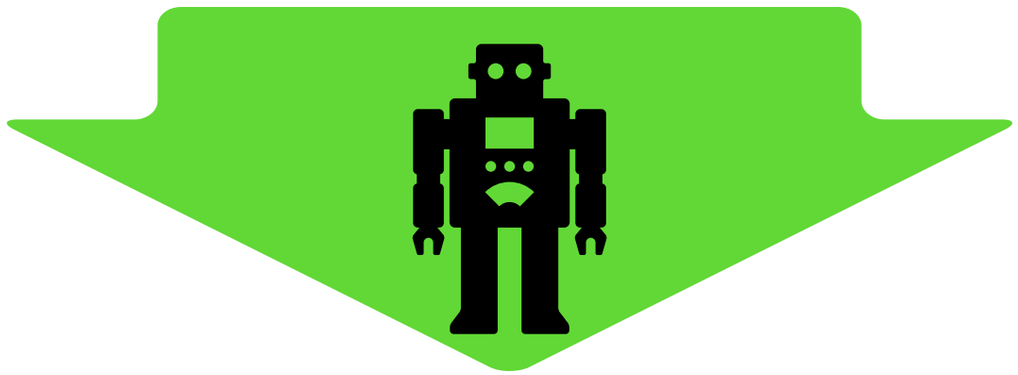
Req

$\forall i. 4 * weeks[i] = months[i]$

$\forall i. 12 * months[i] = years[i]$

Key insight: move hole inside loop

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

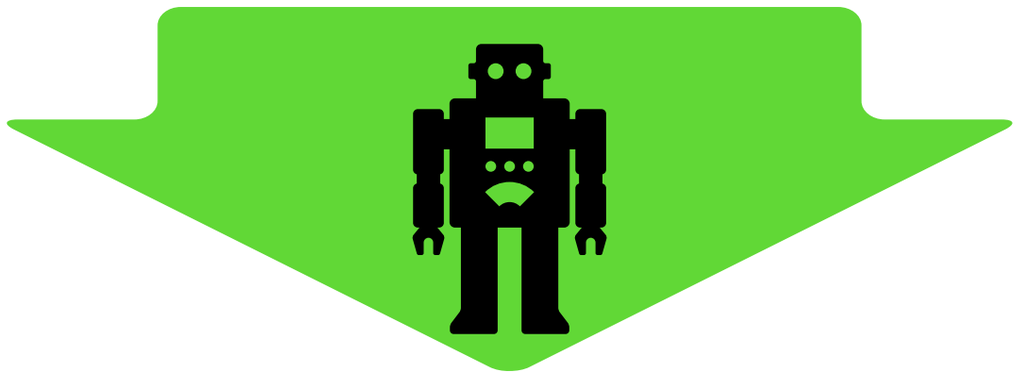
```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
}
for (let i = 0; i < weeks.length; ++i) {
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Key insight: move hole inside loop

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

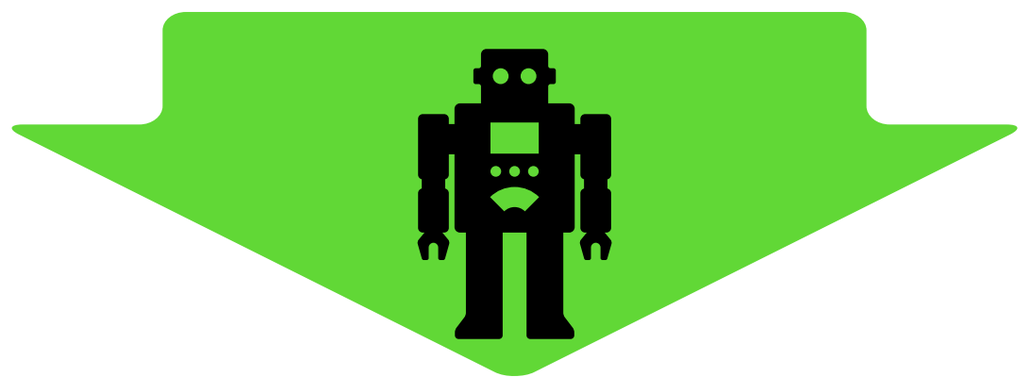
```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Simplification: No quantification!!

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

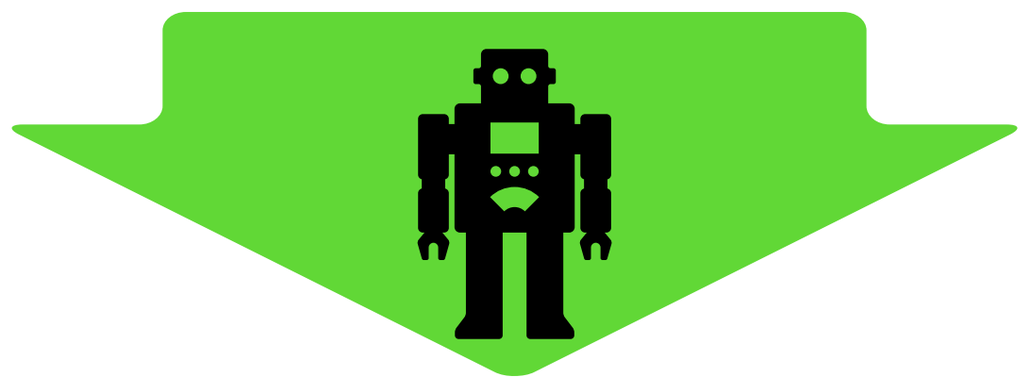
```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Simplification: No quantification!!

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

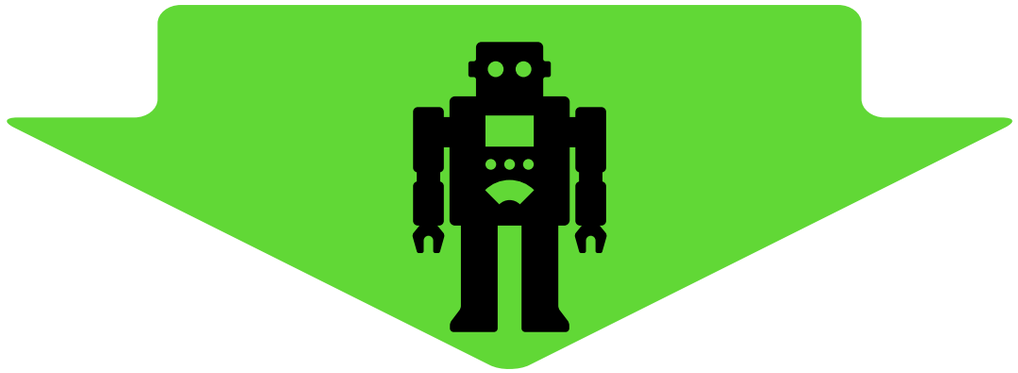
```
for (let i = 0; i < weeks.length; ++i) {
  weeks[i] = weeks[i] * scale;
  months[i] = ??(scale, i, weeks[i]);
  years[i] = ??(scale, i, weeks[i]);
}
```

Req

$\forall i. 4 * weeks[i] = months[i]$
$\forall i. 12 * months[i] = years[i]$

Simplification: No quantification!!

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$$4 * weeks[i] = months[i]$$
$$12 * months[i] = years[i]$$

```
weeks[i] = weeks[i] * scale;
months[i] = ??(scale, i, weeks[i]);
years[i] = ??(scale, i, weeks[i]);
```

Req

$$4 * weeks[i] = months[i]$$
$$12 * months[i] = years[i]$$

Simplification: No quantification!!

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
```

Targeted Synthesis!

Pre

Req

$4 * weeks[i] = months[i]$

$12 * months[i] = years[i]$

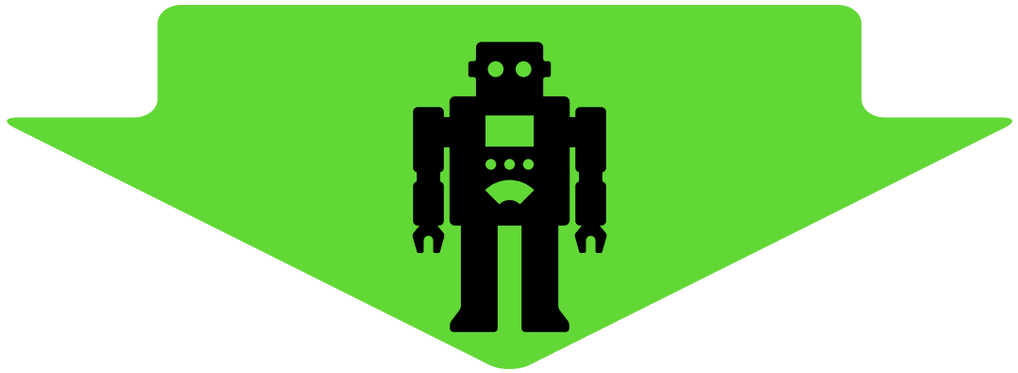
```
weeks[i] = weeks[i] * scale;
months[i] = ??(scale, i, weeks[i]);
years[i] = ??(scale, i, weeks[i]);
```

$4 * weeks[i] = months[i]$

$12 * months[i] = years[i]$

Simplification: No quantification!!

```
function adjustForInflation(scale) {
  for (let i = 0; i < weeks.length; ++i) {
    weeks[i] = weeks[i] * scale;
  }
}
```



Pre

$$4 * weeks[i] = months[i]$$
$$12 * months[i] = years[i]$$

```
weeks[i] = weeks[i] * scale;
months[i] = months[i] * scale;
years[i] = years[i] * scale;
```

Req

$$4 * weeks[i] = months[i]$$
$$12 * months[i] = years[i]$$

Spyder

Separate invariant, imperative programs

Compiler combines with targeted synthesis

Spyder

Automatic Invariant maintenance

Se

Compiler combines with targeted synthesis

Imperative Language

```
for (w: int) with idx in (weeks) {
  if (idx == i) {
    w = val;
  }
}
```

Imperative Language

```
for (w: int) with idx in (weeks) {
  if (idx == i) {
    w = val;
  }
}
```

Loops and Arrays

Imperative Language

```
for (w: int) with idx in (weeks) {
  if (idx == i) {
    w = val;
  }
}
```

In-out iterators

Imperative AST

$$
\begin{array}{lcl}
\mathsf{Block} & ::= & \mathsf{Stmt}_1 \ldots \mathsf{Stmt}_n \\
\mathsf{Stmt} & ::= & \mathsf{Assign} \quad \mathsf{v} \quad \mathsf{Expr} \\
 & \mid & \mathsf{Cond} \quad \mathsf{Expr} \quad \mathsf{Block} \quad \mathsf{Block} \\
 & \mid & \mathsf{For} \quad (\mathsf{v}_1 \ldots \mathsf{v}_n) \quad [\mathsf{v}] \quad (\mathsf{v}_1 \ldots \mathsf{v}_n) \quad \mathsf{Block} \\
 & \mid & \mathsf{VarDecl} \quad (\mathsf{v}, \mathsf{Ty}) \\
\mathsf{Expr} & ::= & \mathsf{v} \mid \mathsf{i} \mid \mathsf{true} \mid \mathsf{false} \\
 & \mid & \mathsf{BinOp} \quad \mathsf{bop} \quad \mathsf{Expr} \quad \mathsf{Expr} \\
 & \mid & \mathsf{Index} \quad \mathsf{Expr} \quad \mathsf{Expr} \\
 & \mid & \mathsf{UnOp} \quad \mathsf{uop} \quad \mathsf{Expr} \\
\mathsf{bop} & ::= & +, \quad \times, \quad \%, \quad \Longrightarrow, \quad \Longleftrightarrow, \ldots \\
\mathsf{uop} & ::= & \neg, \quad !
\end{array}
$$

Invariant Language

```
foreach (week, month) in (weeks, months) {
  week = 4 * month
}
```

Invariant Language

```
foreach (week, month) in (weeks, months) {
  week = 4 * month
}
```

Elementwise Quantification

Invariant AST

$$
\begin{array}{lcl}
\mathsf{Inv} & ::= & \mathsf{ForEach}\ \ (\mathsf{v}_1 \ldots \mathsf{v}_n)\ \ [\mathsf{v}]\ \ (\mathsf{v}_1 \ldots \mathsf{v}_n)\ \ \mathsf{Inv} \\
 & | & \mathsf{BaseExpr}\ \ \mathsf{Expr}
\end{array}
$$

Compilation Algorithm

Language Design Goals

Programmer thinks about invariants once

Compiler automatically handles invariants

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Format

Invariants

Code

Explanatory Caption

Problem/Result

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

X ~ Y

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

X ~ Y

Y edited

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

X ~ Y

Y edited

```
x = ??(x, y);
```

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

X, Y in scope

```
x = ??(x, y);
```

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

```
x = ??(x, y);
```

Simple Example

```
x > 2 * y
```

```
y = y + 1;
```

```
x = x + 2;
```

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Extending Loops

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int) in (weeks) {
  w = w * 1.05;
}
```

Extending Loops

```
foreach (w, m) in (weeks, months) {
  w = 4 * m
}
```

```
for (w: int) in (weeks) {
  w = w * 1.05;
}
```

weeks ~ months

Extending Loops

```
foreach (w, m) in (weeks, months) {
  w = 4 * m
}
```

weeks ~ months

```
for (w: int) in (weeks) {
  w = w * 1.05;
}
```

need months

Extending Loops

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int) in (weeks) {
  w = w * 1.05;
}
```

weeks ~ months

need months

Extending Loops

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

weeks ~ months

```
for (w: int, m: int) in (weeks, months){
  w = w * 1.05;
}
```

need months

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Simplifying Invariants

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int, m: int) in (weeks, months){
  w = w * 1.05;
}
```

Simplifying Invariants

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int, m: int) in (weeks, months){
  w = w * 1.05;
}
```

Key insight: foreach body <=> for invariant

Simplifying Invariants

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int, m: int) in (weeks, months){
  w = w * 1.05;
}
```

Key insight: foreach body <=> for invariant

```
m = ??(m, w);
```

Simplifying Invariants

```
foreach (w, m) in (weeks, months){
  w = 4 * m
}
```

```
for (w: int, m: int) in (weeks, months){
  w = w * 1.05;
}
```

Key insight: foreach body <=> for invariant

```
m = m * 1.05;
```

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Compilation Algorithm

1) bring necessary variables into scope

2) specialize invariants to loops

3) ??? (find template)

4) profit (synthesize)

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Budgeting Example

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$

$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$

$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Budgeting Example

Weekly Monthly Yearly

-40

25

75

-45 -180 -2160

Color of Odd Rows

Color of Even Rows

Invariants

$\forall i.i \mod 2 = 0 \implies bkg[i] = even$

$\forall i.i \mod 2 = 1 \implies bkg[i] = odd$

$= red$

$= black$

Need to synthesize loops, conditionals

Budgeting Example

Weekly Monthly Yearly

-40

25

75

-45 -180 -2160

Color of Odd Rows

Color of Even Rows

Invariants

$\forall i.i \mod 2 = 0 \implies bkg[i] = even$

$\forall i.i \mod 2 = 1 \implies bkg[i] = odd$

$= red$

$= black$

Need to synthesize **loops**, conditionals

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

bkg ~ bkgColors

bkg edited

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

bkg ~ bkgColors

bkg edited

```
for (color: int) in (bkgColors){
  color = ??(bkg, blue);
}
```

Loops

That's the skeleton…
What about the invariant?

```
for (color: int) in (bkgColors){
  color = ??(bkg, blue);
}
```

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

```
for (color: int) in (bkgColors){
  color = ??(bkg, blue);
}
```

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

Use foreach spec for body

```
for (color: int) in (bkgColors){
  color = ??(bkg, blue);
}
```

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

```
for (color: int) in (bkgColors){
  color = ??(bkg, blue);
}
```

Synthesizing Loops

```
foreach (color) in (bkgColors) {
  color = bkg
}
```

```
bkg = blue;
```

```
for (color: int) in (bkgColors){
  color = blue;
}
```

Budgeting Example

Invariants

Weekly Monthly Yearly

-40

25

750

-45 -180 -2160

Color of Odd Rows

Color of Even Rows

$\forall i.i \mod 2 = 0 \implies bkg[i] = even$

$\forall i.i \mod 2 = 1 \implies bkg[i] = odd$

$= red$

$= black$

Need to synthesize **loops**, conditionals

Budgeting Example

Invariants

Weekly Monthly Yearly

-40

25

750

-45 -180 -2160

Color of Odd Rows

Color of Even Rows

$\forall i.i \mod 2 = 0 \implies bkg[i] = even$

$\forall i.i \mod 2 = 1 \implies bkg[i] = odd$

$= red$

$= black$

Need to synthesize loops, **conditionals**

Synthesizing Conditionals

```
(idx % 2 = 0) ==> color = even
```

```
even = red;
```

Synthesizing Conditionals

```
idx % 2 = 0) => color = even
```

```
even = red;
```

Implications as conditionals

Synthesizing Conditionals

```
idx % 2 = 0) ==> color = even
```

```
even = red;
```

Implications as conditionals

```
if (idx % 2 == 0) {
   color = ??(even, red, idx);
}
```

Synthesizing Conditionals

```
(idx % 2 = 0) ==> color = even
```

```
even = red;
```

```
if (idx % 2 == 0) {
   color = ??(even, red, idx);
}
```

Synthesizing Conditionals

```
(idx % 2 = 0) ==> color = even
```

```
even = red;
```

```
if (idx % 2 == 0) {
   color = even;
}
```

Case studies*

Game of Life (1d, 2d)

Budgeting calculator

Split expenses spreadsheet

*

Game of Life

Invariants

$$\forall i.model[i] \implies view[i] = black$$
$$\forall i.!model[i] \implies view[i] = white$$
$$running \iff start\_enabled$$

Code

- State transition
- Interactive display
- Start/Stop toggle

*

Budgeting

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

Invariants

$$\forall i. 4 * weeks[i] = months[i]$$
$$\forall i. 12 * months[i] = years[i]$$
$$\forall i. weeks[i] < 0 \implies font[i] = red$$
$$\forall i. weeks[i] >= 0 \implies font[i] = black$$
$$\forall i. i \mod 2 = 0 \implies bkg[i] = even$$
$$\forall i. i \mod 2 = 1 \implies bkg[i] = odd$$

Code

- Interactive cells
- Inflation
- Backgrounds

*

Split expenses

Weekly	Monthly	Yearly
-400	-1600	-19200
25	100	1200
750	3000	36000
-45	-180	-2160

Color of Odd Rows

Color of Even Rows

*

Invariants

$$\forall i. 2 * mid[i] = me[i] + you[i]$$
$$\forall i. debt[i] = mid[i] - you[i]$$
$$\forall i. debt[i] < 0 \implies color[i] = black$$
$$\forall i. debt[i] >= 0 \implies color[i] = red$$

Code

- Interactive cells
- Backgrounds

Outline

- **Data invariant calculus**
- Extensions
- Invariants for Security
- GUI invariants

Outline

- Data invariant calculus
- **Extensions**
- Invariants for Security
- GUI invariants

Formalisms

Formalisms

$$
\begin{array}{lcl}
\kappa & \in & \mathcal{P}(\mathsf{Inv}) \\
\Downarrow_i & \in & \mathcal{P}(\mathsf{Inv}) \rightarrow \mathbf{Expr} \\
\Downarrow_s & \in & (\mathcal{P}(\mathsf{Inv}), \mathsf{Stmt}) \rightarrow \mathbf{Stmt} \\
\mathsf{body} & \in & [\mathsf{Vars}] \rightarrow \mathsf{Vars} \rightarrow [\mathsf{Vars}] \rightarrow \mathsf{Inv} \rightarrow \mathsf{Inv} \\
\mathtt{iters} & \in & [\mathsf{Vars}] \rightarrow \mathsf{Vars} \rightarrow [\mathsf{Vars}] \rightarrow (\mathbf{Expr}, \mathbf{blk}, \mathbf{blk}, \mathbf{blk})
\end{array}
$$

(a) Declarations for functions used in translation from SPYDER to IMP-VERIF.

$$
\text{Trans-Cond}\ \frac{\kappa \Downarrow_i \mathbf{i} \qquad \kappa, \mathsf{blk}_t \Downarrow_s \mathbf{blk}_t \qquad \kappa, \mathsf{blk}_f \Downarrow_s \mathbf{blk}_f}{\kappa, \mathsf{Cond}\ \mathsf{e}\ \mathsf{blk}_t\ \mathsf{blk}_f \Downarrow_s \mathbf{Enf}\ \mathbf{i}\ ;\ \mathbf{Cond}\ \mathbf{e}\ \mathbf{blk}_t\ \mathbf{blk}_f\ ;}
$$

$$
\text{Trans-For}\ \frac{\kappa \Downarrow_i \mathbf{i} \quad \kappa' \Downarrow_i \mathbf{i}' \qquad \begin{array}{c}\kappa' = \mathsf{map}\,(\mathsf{body}\ \mathsf{v}_i\ \mathsf{idx}\ \mathsf{arr}_i)\ \kappa \\ (\mathbf{e}, \mathbf{blk}_{let}, \mathbf{blk}_{in}, \mathbf{blk}_{out}) = \mathtt{iters}\ \mathsf{v}_i\ \mathsf{idx}\ \mathsf{arr}_i\end{array} \qquad \kappa', \mathsf{blk} \Downarrow_s \mathbf{blk}}{\kappa, \mathsf{For}\ (\mathsf{v}_i)\ \mathsf{idx}\ (\mathsf{arr}_i)\ \mathsf{blk} \Downarrow_s \mathbf{Enf}\ \mathbf{i};\ \mathbf{blk}_{let};\ \mathbf{While}\ \mathbf{e}\ \mathbf{i}\ (\mathbf{blk}_{in};\ \mathbf{Assume}\ \mathbf{i}';\ \mathbf{blk};\ \mathbf{Enf}\ \mathbf{i}';\ \mathbf{blk}_{out})}
$$

(b) Semantic definitions for translating conditional and loop statements in SPYDER to statements in IMP-VERIF.

Figure 7. Translation Semantics between SPYDER and IMP-VERIF.

Formalisms

$\kappa \in \mathcal{P}(\mathsf{Inv})$

Interesting
Translation Rule

$$\text{Trans-Cond}\ \frac{}{\kappa, \mathsf{Cond}\ \mathsf{e}\ \mathsf{blk}_t\ \mathsf{blk}_f \Downarrow_s \mathbf{Enf}\ \mathbf{i}\ ;\ \mathbf{Cond}\ \mathbf{e}\ \mathbf{blk}_t\ \mathbf{blk}_f\ ;}$$

$$\text{Trans-For}\ \frac{\kappa \Downarrow_i \mathbf{i} \quad \kappa' \Downarrow_i \mathbf{i}' \quad \begin{array}{c}\kappa' = \mathsf{map}\,(\mathsf{body}\ \mathsf{v}_i\ \mathsf{idx}\ \mathsf{arr}_i)\ \kappa \\ (\mathbf{e}, \mathbf{blk}_{let}, \mathbf{blk}_{in}, \mathbf{blk}_{out}) = \mathtt{iters}\ \mathsf{v}_i\ \mathsf{idx}\ \mathsf{arr}_i\end{array} \quad \kappa', \mathsf{blk} \Downarrow_s \mathbf{blk}}{\kappa, \mathsf{For}\ (\mathsf{v}_i)\ \mathsf{idx}\ (\mathsf{arr}_i)\ \mathsf{blk} \Downarrow_s \mathbf{Enf}\ \mathbf{i};\ \mathbf{blk}_{let};\ \mathbf{While}\ \mathbf{e}\ \mathbf{i}\ (\mathbf{blk}_{in};\ \mathbf{Assume}\ \mathbf{i}';\ \mathbf{blk};\ \mathbf{Enf}\ \mathbf{i}';\ \mathbf{blk}_{out})}$$

(b) Semantic definitions for translating conditional and loop statements in SPYDER to statements in IMP-VERIF.

Figure 7. Translation Semantics between SPYDER and IMP-VERIF.

Formalisms

What we need

- Formal problem definition
- Full translation semantics
- Proof of correctness

Formal Correctness

Problem:

Correctness:

Formal Correctness

Problem:

Given a program with arrays and invariants…

Correctness:

Formal Correctness

Problem:

Given a program with arrays and invariants…

Correctness:

We find a program with augmented semantics…

Formal Correctness

Problem:

Given a program with arrays and invariants

Thank you, Reviewer A!!!

Correctness:

We find a program with augmented semantics…

Stronger Invariants

$$\forall i, j.\ \ i < j \implies sorted[i] < sorted[j]$$

Consider Sortedness…

Stronger Invariants

$$\forall i, j.\ \ i < j \implies sorted[i] < sorted[j]$$

$$\begin{array}{lcl} \mathsf{Inv} & ::= & \mathsf{ForEach}\ \ (\mathsf{v}_1 \ldots \mathsf{v}_n)\ \ [\mathsf{v}]\ \ (\mathsf{v}_1 \ldots \mathsf{v}_n)\ \ \mathsf{Inv} \\ & | & \mathsf{BaseExpr}\ \ \mathsf{Expr} \end{array}$$

Stronger Invariants

$$\forall i, j.\ \ i < j \implies sorted[i] < sorted[j]$$

$$\begin{array}{lcl} \text{Inv} & ::= & \text{ForEach}\ \ (\text{v}_1 \ldots \text{v}_n)\ \ [\text{v}]\ \ (\text{v}_1 \ldots \text{v}_n)\ \ \text{Inv} \\ & | & \text{BaseExpr}\ \ \text{Expr} \end{array}$$

Limited to
“maplike” properties

Stronger Invariants

$$\forall i, j.\ \ i < j \implies sorted[i] < sorted[j]$$

$$\begin{array}{lcl} \text{Inv} & ::= & \text{ForEach}\ \ (\text{v}_1 \ldots \text{v}_n)\ \ [\text{v}]\ \ (\text{v}_1 \ldots \text{v}_n)\ \ \text{Inv} \\ & | & \text{BaseExpr}\ \ \text{Expr} \end{array}$$

Cannot express :(

Stronger Invariants

Key insight:
change of perspective!!

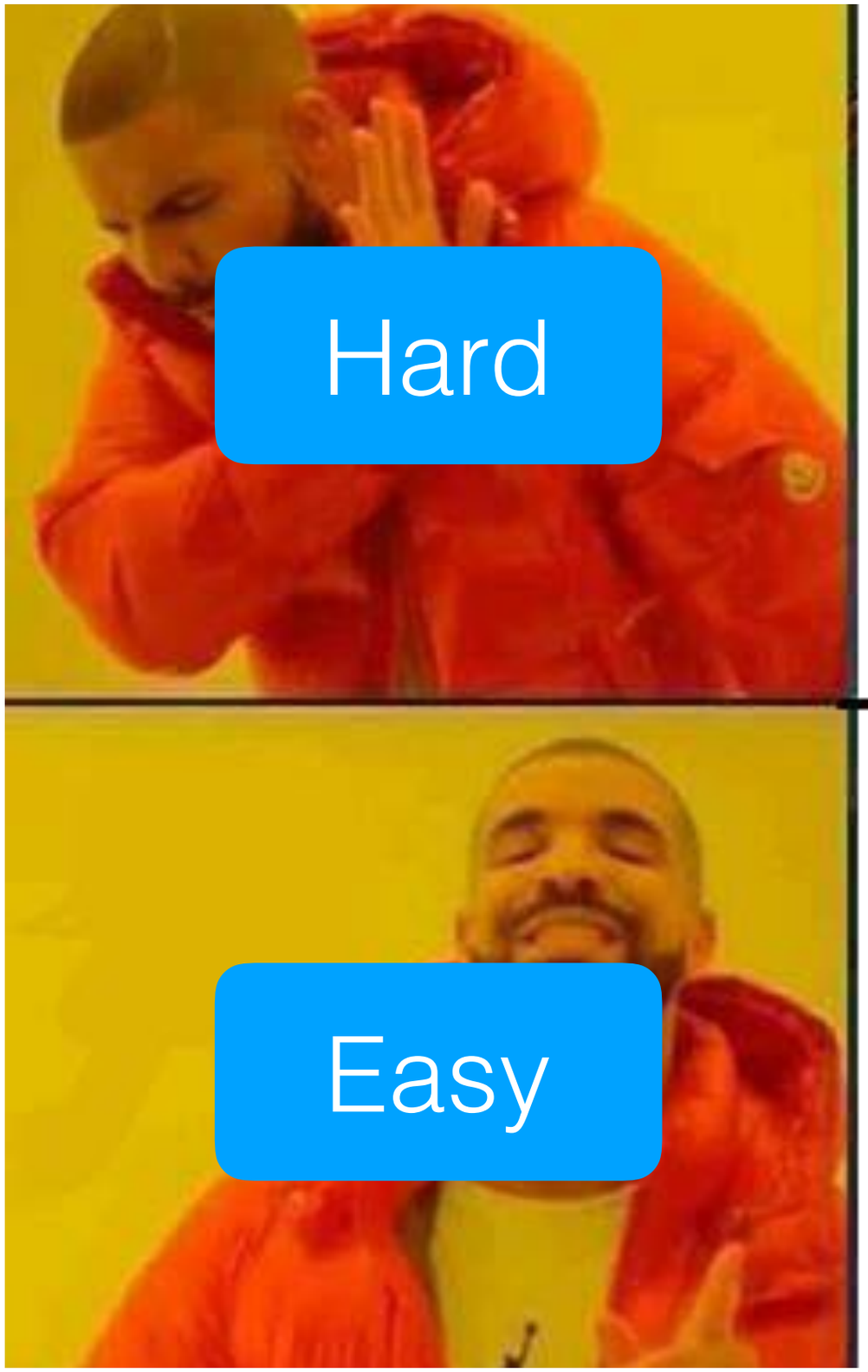
Cannot express :(

Stronger Invariants

$$\forall i, j.\ \ i < j \implies sorted[i] < sorted[j]$$

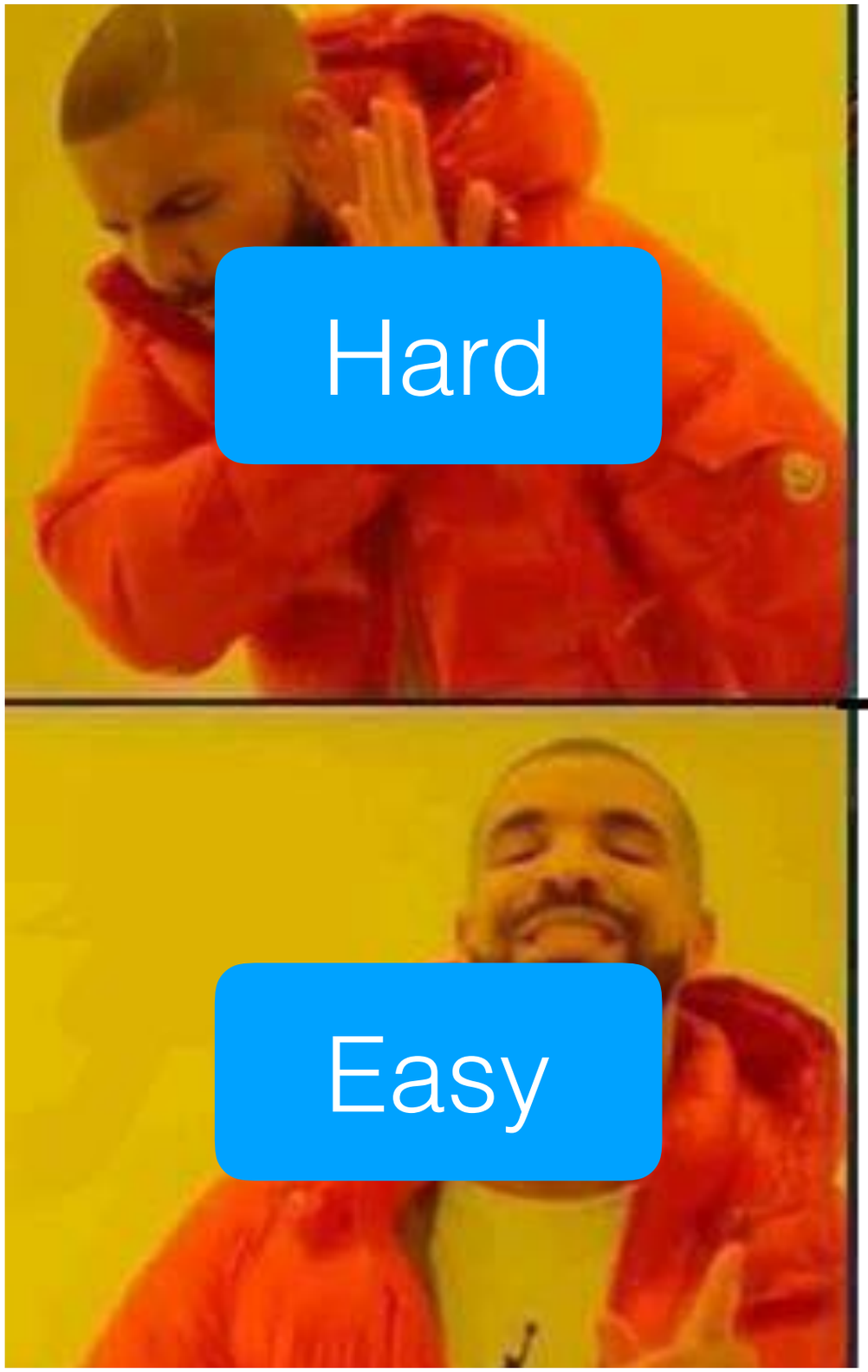
$$\forall i, j. \;\; i < j \implies sorted[i] < sorted[i+1]$$

$$\forall i. \;\; sorted[i] < sorted[i+1]$$



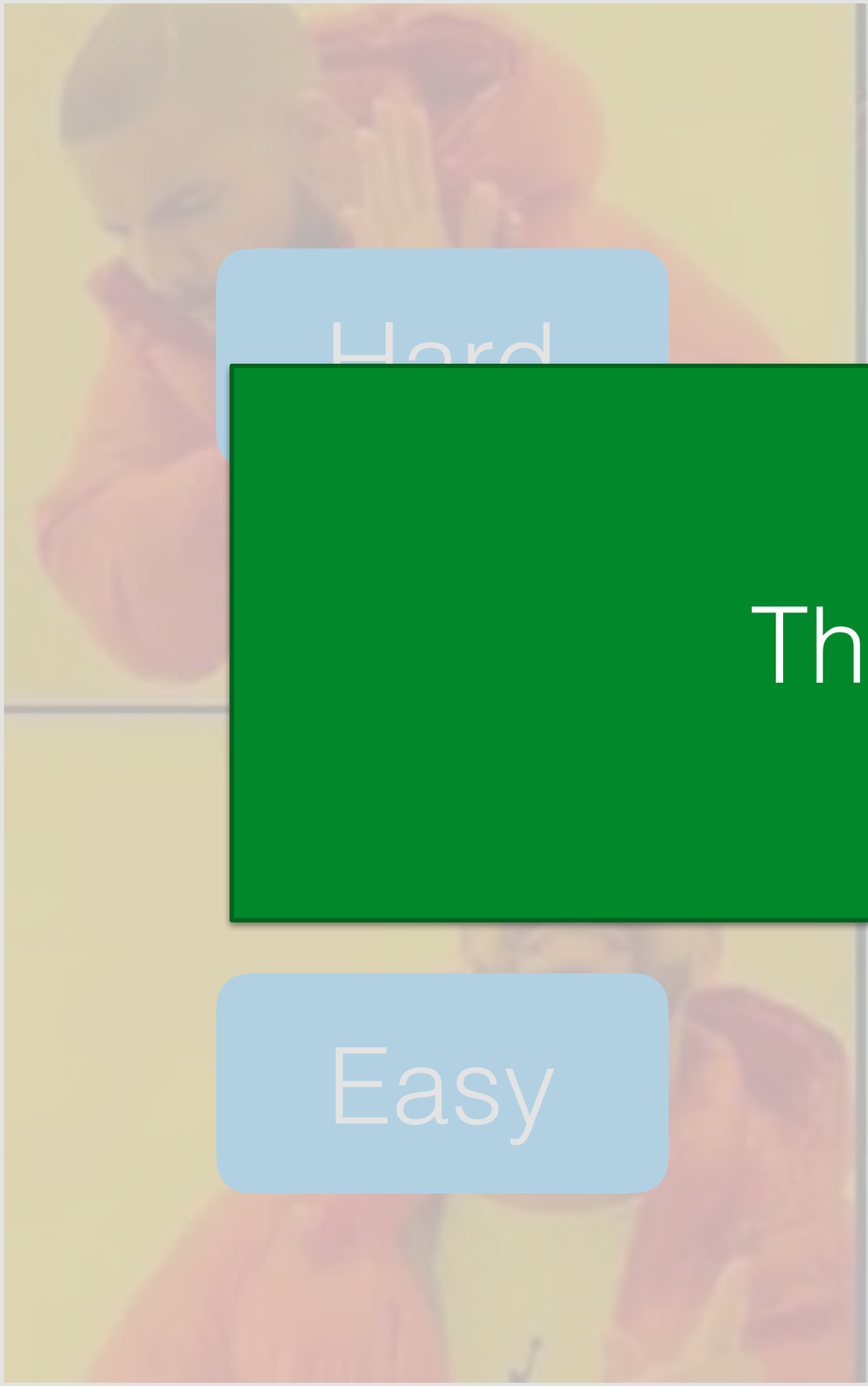
$$\forall i, j. \;\; i < j \implies sorted[i] < sorted[i+1]$$

$$\forall i. \;\; sorted[i] < sorted[i+1]$$



$$\forall i, j. \quad P(i, j)$$

$$\forall i. P(i, i+1)$$



Hard

$$\forall i, j, \; P(i, j)$$

Thanks Nadia!!

Easy

$$\forall i. P(i, i+1)$$

Synthesizing Insert

```
adjacent (lo, hi) in (sorted) {
  lo < hi
}
```

Synthesizing Insert

```
adjacent (lo, hi) in (sorted) {
  lo < hi
}
```

```
sorted = sorted + newElem
```

Synthesizing Insert

```
adjacent (lo, hi) in (sorted) {
  lo < hi
}
```

```
sorted = sorted + newElem
```

```
sorted = foldl adder ([], false) sorted
adder (es, d) next = ...
```

Synthesizing Insert

```
adjacent (lo, hi) in (sorted) {
  lo < hi
}
```

```
sorted = sorted + newElem
```

But in imperative language

```
sorted = foldl adder ([], false) sorted
adder (es, d) next = ...
```

Implications

- Not clear — need to adjust at least
  - 1) loop translation
  - 2) synthesis
- Formal correctness would help!!

Outline

- Data invariant calculus
- **Extensions**
- Invariants for Security
- GUI invariants

Outline

- Data invariant calculus
- Extensions
- **Invariants for Security**
- GUI invariants

Security is really hard

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

jtsarracino@gmail.com

pwned?

Adobe: In October 2013, 153 million Adobe accounts were breached with each containing an internal ID, username, email, *encrypted* password and a password hint in plain text. The password cryptography was poorly done and many were quickly resolved back to plain text. The unencrypted hints also disclosed much about the passwords adding further to the risk that hundreds of millions of Adobe customers already faced.

Compromised data: Email addresses, Password hints, Passwords, Usernames

KICKSTARTER

Kickstarter: In February 2014, the crowdfunding platform Kickstarter announced they'd suffered a data breach. The breach contained almost 5.2 million unique email addresses, usernames and salted SHA1 hashes of passwords.

Compromised data: Email addresses, Passwords

LinkedIn: In May 2016, LinkedIn had 164 million email addresses and passwords exposed. Originally hacked in 2012, the data remained out of sight until being offered for sale on a dark market site 4 years later. The passwords in the breach were stored as SHA1 hashes without salt, the vast majority of which were quickly cracked in the days following the release of the data.

Compromised data: Email addresses, Passwords

Ticketfly: In May 2018, the website for the ticket distribution service Ticketfly was defaced by an attacker and was subsequently taken offline. The attacker allegedly requested a ransom to share details of the vulnerability with Ticketfly but did not receive a reply and subsequently posted the breached data online to a publicly accessible location. The data included over 26 million unique email addresses along with names, physical addresses and phone numbers. Whilst there were no

EQUIFAX®

140 MILLION PEOPLE

NAMES

DRIVERS LICENSES

SOCIAL SECURITY NUMBERS

BIRTHDATES

ADRESSES

Facebook Network is Breached,
Putting 50 Million Users' Data at Risk

TECH

Facebook Faces Potential $1.63 Billion Fine in Europe Over Data Breach

Privacy watchdog looks into whether social network violated European's Union new privacy law

Hot off the presses!! September 28

Manual Security Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant:

"All urls are in the urlset"

Manual Security Invariants

```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

Invariant:

"All urls are in the urlset"

"All connections use the urlset"

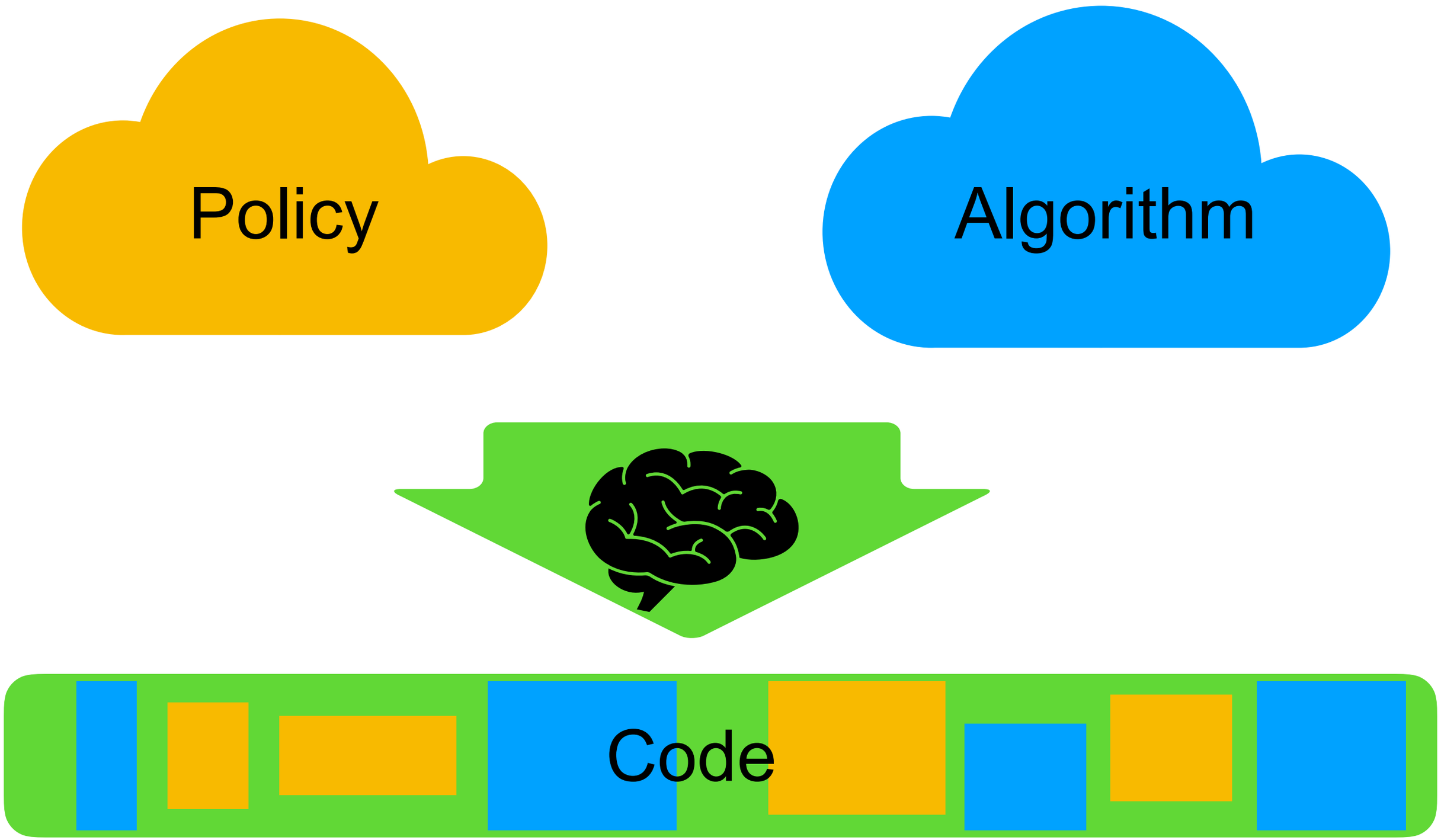
Manual Security Invariants

Invariant:

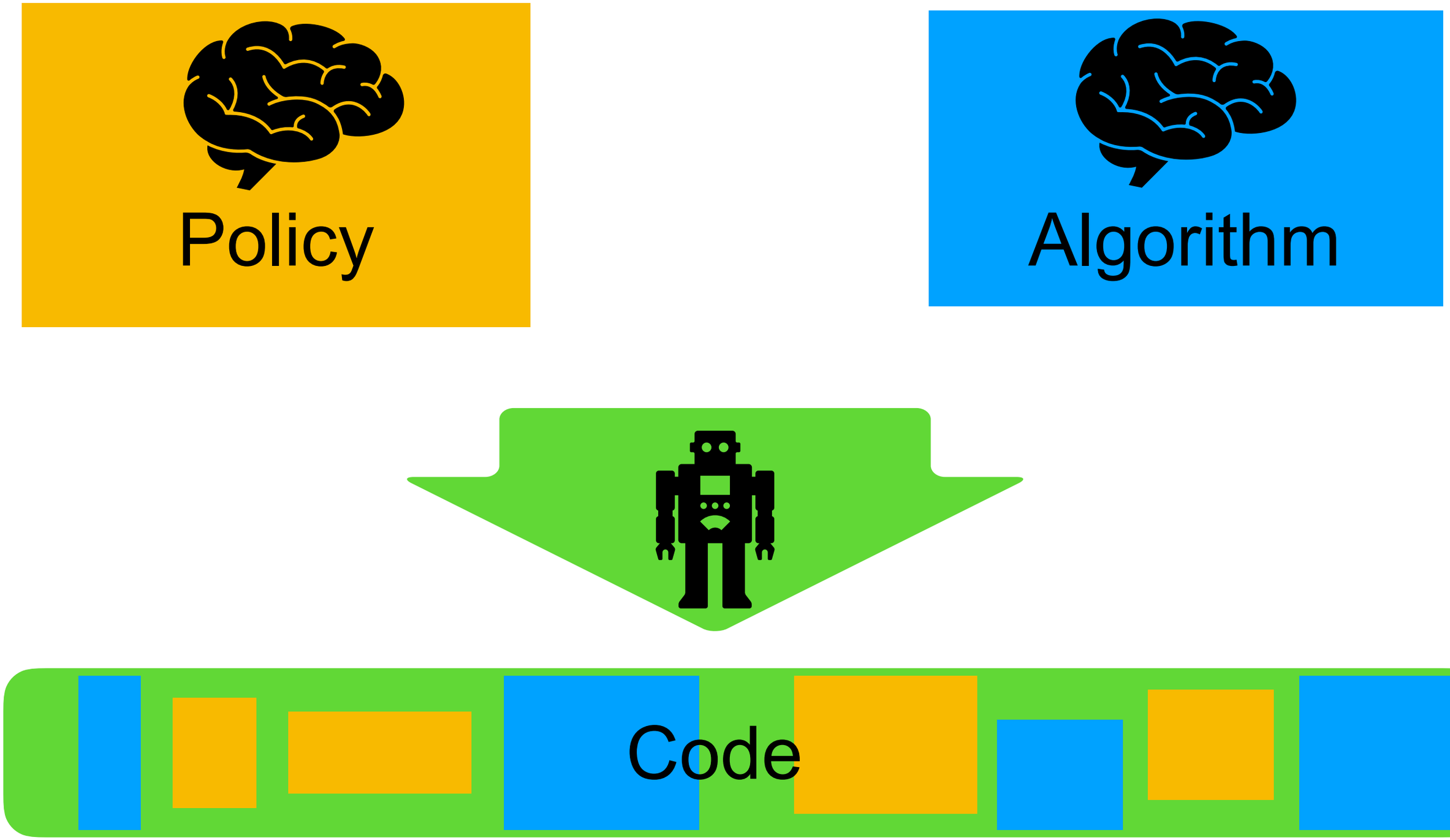
"All urls are in the urlset"

"All connections use the urlset"

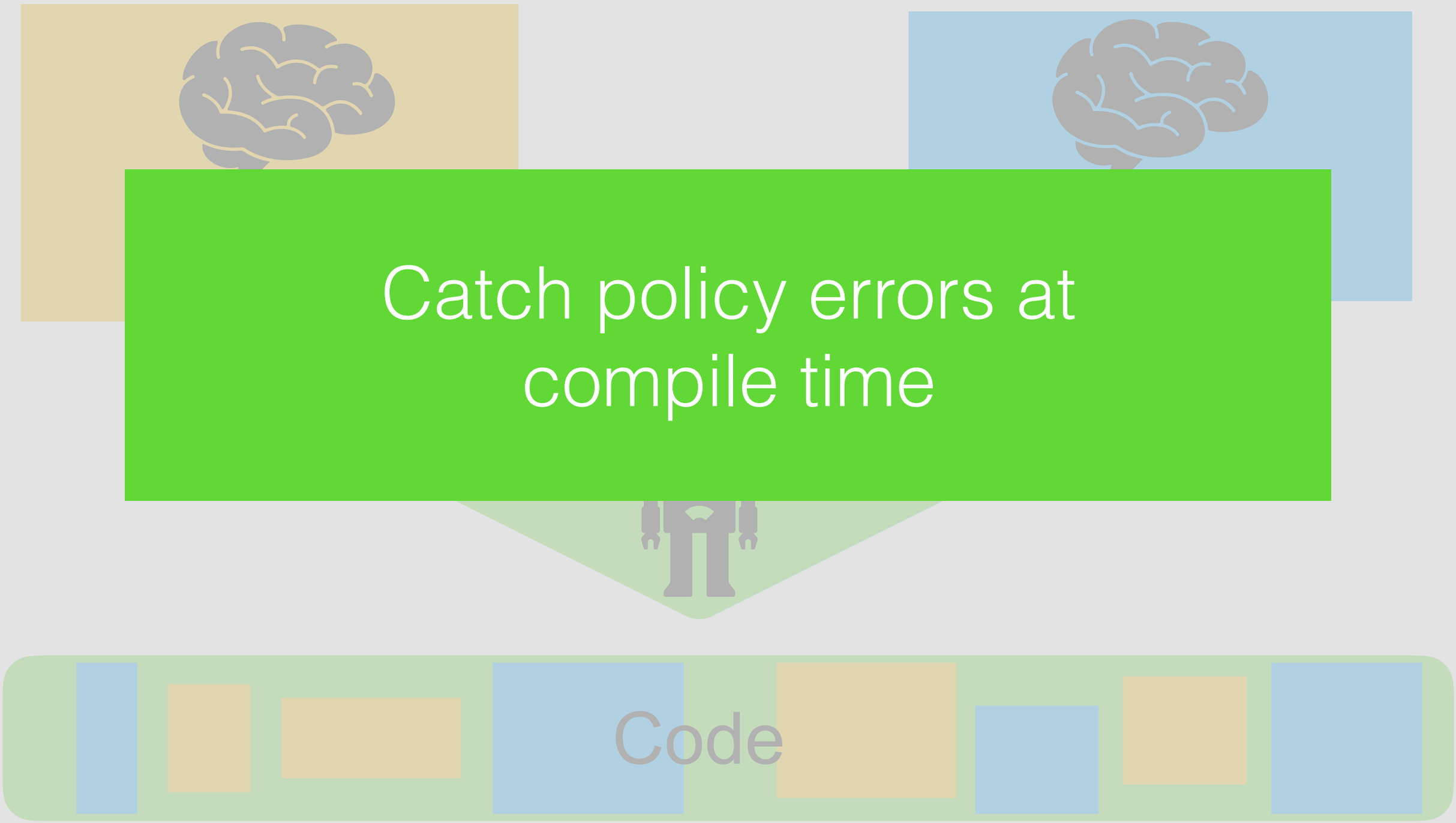
Manual Security Programming



Security Synthesis



Security Synthesis



Security Policy Spectrum

Easy Hard Insane

Security Policy Spectrum

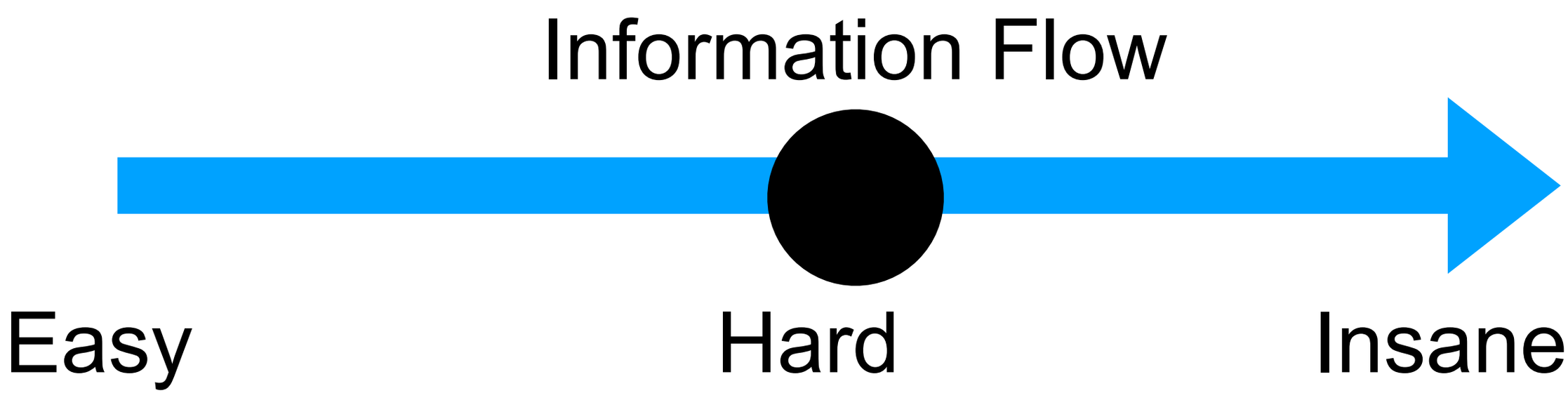
“all urls are safe”

Permissions

Easy Hard Insane

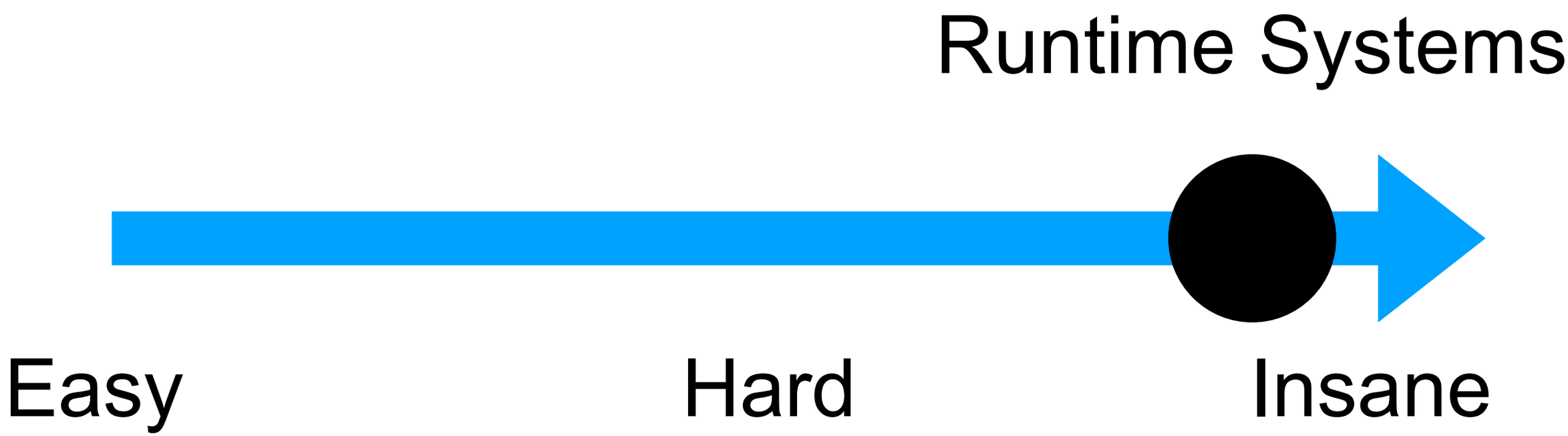
Security Policy Spectrum

"the secret is not leaked"



Security Policy Spectrum

"no buffers overflow"



Security Policy Spectrum

“all urls are safe”

Permissions

Easy Hard Insane

Permission Invariants

```
foreach (u1, u2) in (Users, Users) {
  u1 can view info(u2) <=>
  u1 in Doctors && u2 in patients(u1)
}
```

```
render(u: User) {
  for (record) in (records) {
    display record
  }
}
```

Permission Invariants

```
foreach (u1, u2) in (Users, Users) {
  u1 can view info(u2) <=>
  u1 in Doctors && u2 in patients(u1)
}
```

```
render(u: User) {
  for (record) in (records) {
    display record
  }
}
```

```
render(u: User) {
  for (record) in (records) {
    if (u in Doctors && patient(record) in patients(u)) {
      display record
    }
  }
}
```

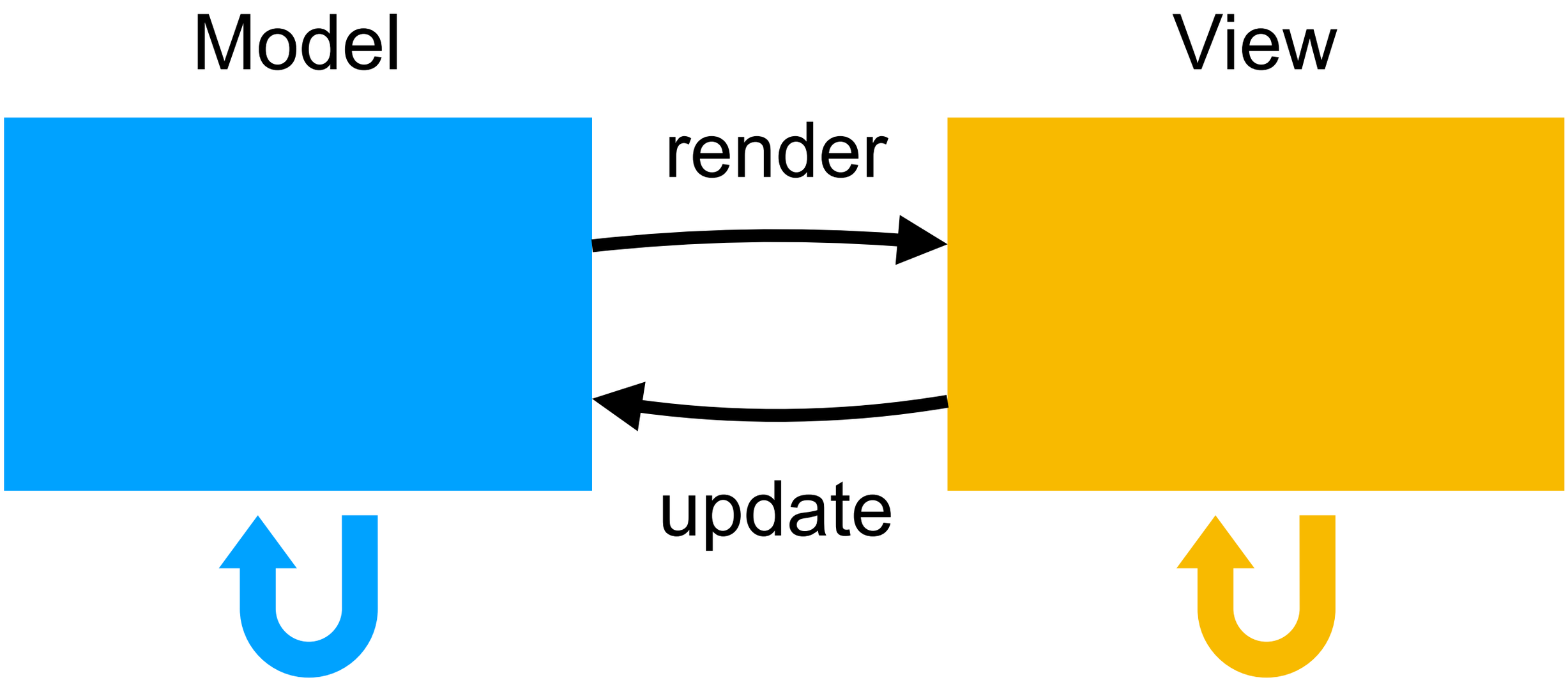
Outline

- Data invariant calculus
- Extensions
- **Invariants for Security**
- GUI invariants

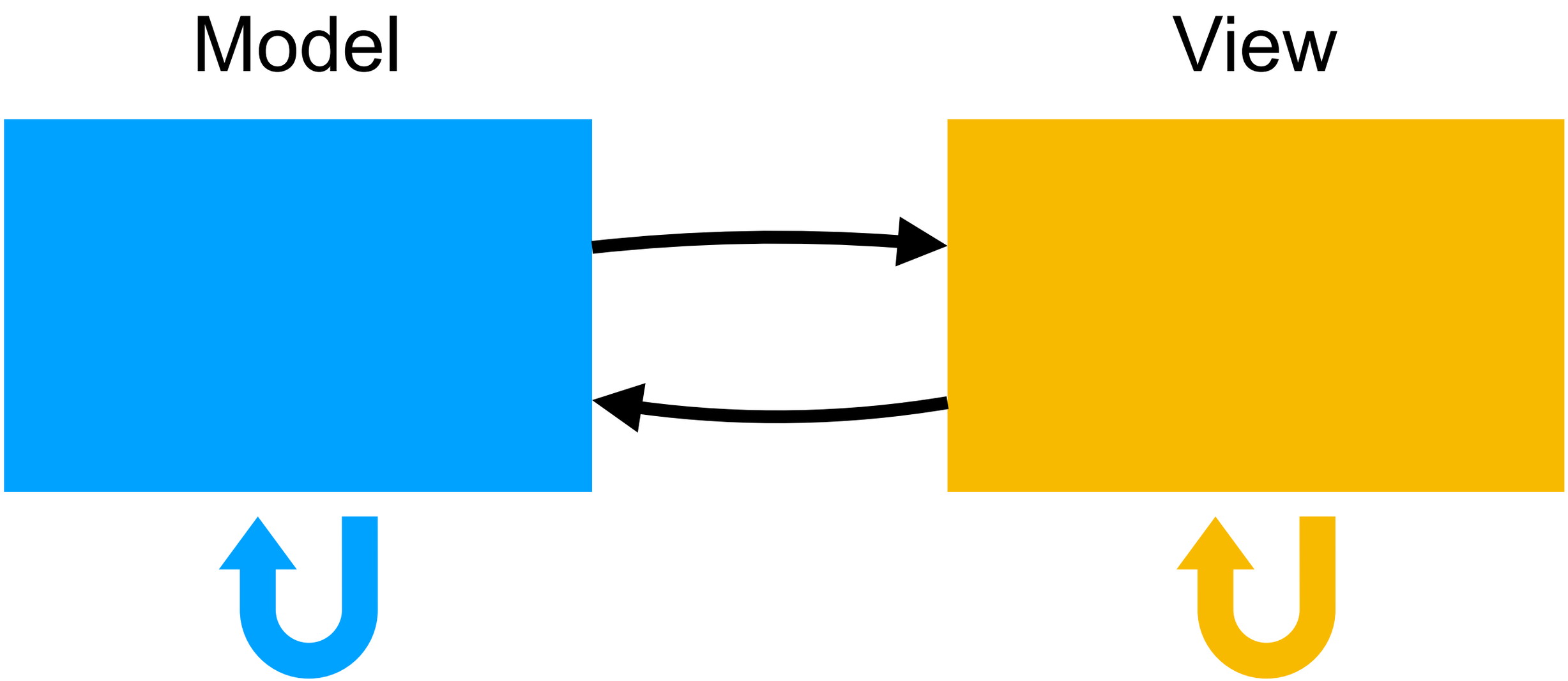
Outline

- Data invariant calculus
- Extensions
- Invariants for Security
- **GUI invariants**

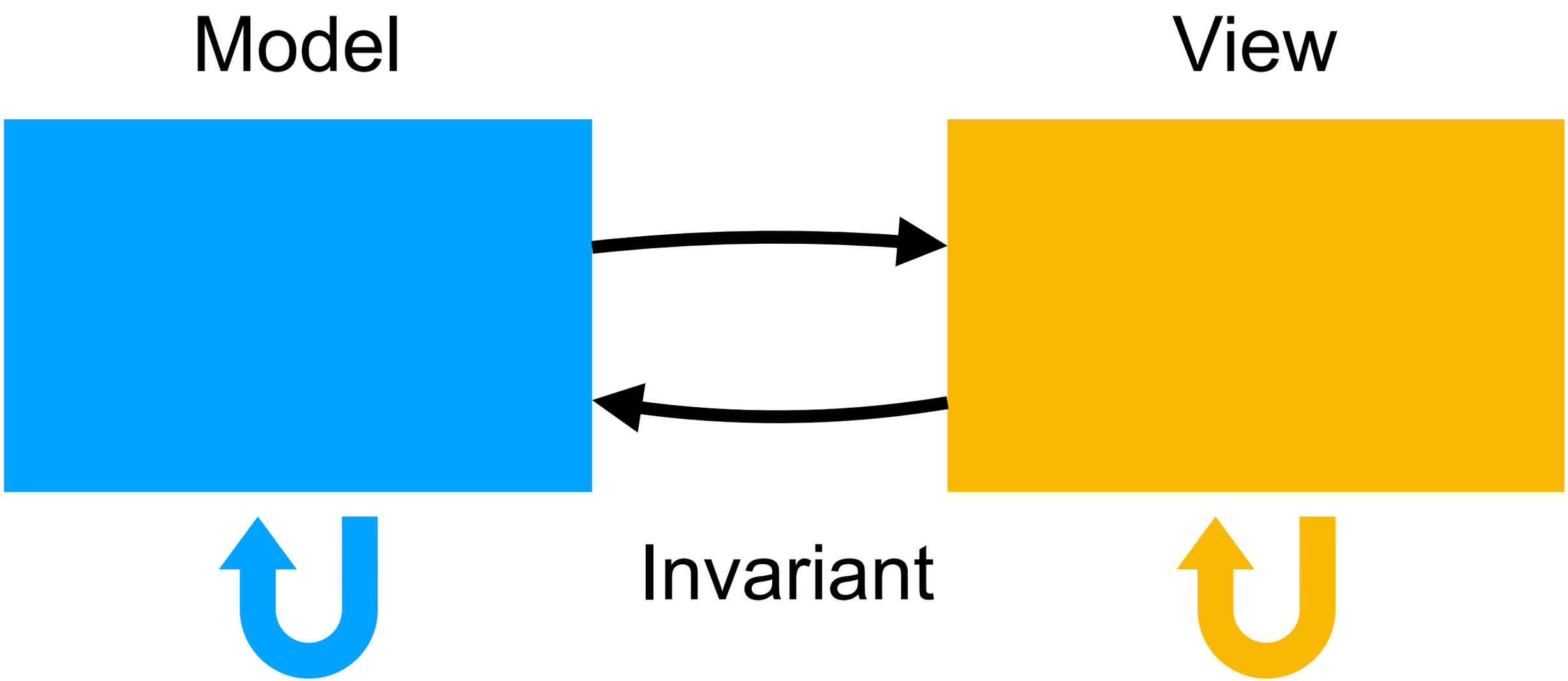
Model-View Synchronization



Synchronization as Invariants



Synchronization as Invariants



GUI Invariants

“all text has

sufficient contrast”

“append
post_content to the
end of tweet_body”

GUI Invariants

“all text has

sufficient contrast”

“append
post_content to the
end of tweet_body”

Inspired by Cassius, PLDI ’18:
Verifying that Web Pages have Accessible Layout

GUI Invariants

```
foreach txtBlock in texts(document) {
  foreach block in blocks(document) {
    block isBehind txtBlock ==>
       goodContrast(txtBlock, block)
  }
}
```

```
tweet_body.appendChild(post_content);
```

GUI Invariants

```
foreach txtBlock in texts(document) {
  foreach block in blocks(document) {
    block isBehind txtBlock ==>
       goodContrast(txtBlock, block)
  }
}
```

```
tweet_body.appendChild(post_content);
```

```
t1 = text(txtBlock[0:mid], "blue");
t2 = text(txtBlock[mid:end], "black");
t3 = text(txtBlock[end:], "white");
```

```
spn.appendChild(t1)
   .appendChild(t2)
   .appendChild(t3);
tweet_body.appendChild(spn);
```

GUI Invariants

```
foreach txtBlock in texts(document) {
  foreach block in blocks(document) {
    block isBehind txtBlock ==>
       goodContrast(txtBlock, block)
  }
}
```

```
tweet_body.appendChild(post_content);
```

Accessibility-agnostic programming!!

```
t1 = text(txtBlock[0:mid], "blue");
t2 = text(txtBlock[mid:end], "black");
t3 = text(txtBlock[end:], "white");
```

```
spn.appendChild(t1)
   .appendChild(t2)
   .appendChild(t3);
tweet_body.appendChild(spn);
```

Outline

- Data invariant calculus
- Extensions
- Invariants for Security
- **GUI invariants**

My thesis proposal

Compiling and generating invariants is useful, novel, and broadly applicable.

I propose a general framework for maintaining invariants