Introduction to Object Technology

25 October 2000

Theo D'Hondt

Programming Technology Lab
Vrije Universiteit Brussel
http://prog.vub.ac.be/~tjdhondt

ftp://dinf.vub.ac.be/pub/TOOLSEE/ObjTech.pdf

Contents

- Definition
- Abstraction
- Polymorphism
- Inheritance
- Frameworks
- Types
- Simulation
- Conduits

Definition

Object orientation

=

data abstraction

+

structured polymorphism

Definition (cont'd)

Object orientation

=

data abstraction

+

structured polymorphism

Contents (cont'd)

- ☑ Definition
- ☐ Abstraction
  - Names
  - Procedures
  - Composition
  - Data abstraction
- ☐ Polymorphism
- ☐ ...

Abstraction: names

```
leap-year?
  if multiple-of? year 4
    if multiple-of? year 100
      if multiple-of? year 400
        true
        false
      true
    false
```

Abstraction: names (cont'd)

```
leap-year?
   if multiple-of? year 4
      if multiple-of? year 100
         if multiple-of? year 400
            true
            false
         true
      false
```

Naming ...

Abstraction: procedures

```
leap-year? year
  if multiple-of? year 4
    if multiple-of? year 100
      if multiple-of? year 400
        true
        false
      true
    false
```

Abstraction: procedures (cont'd)

```
leap-year? year
  if multiple-of? year 4
    if multiple-of? year 100
      if multiple-of? year 400
        true
        false
      true
    false
```

Parametrizing ...

Abstraction: composition

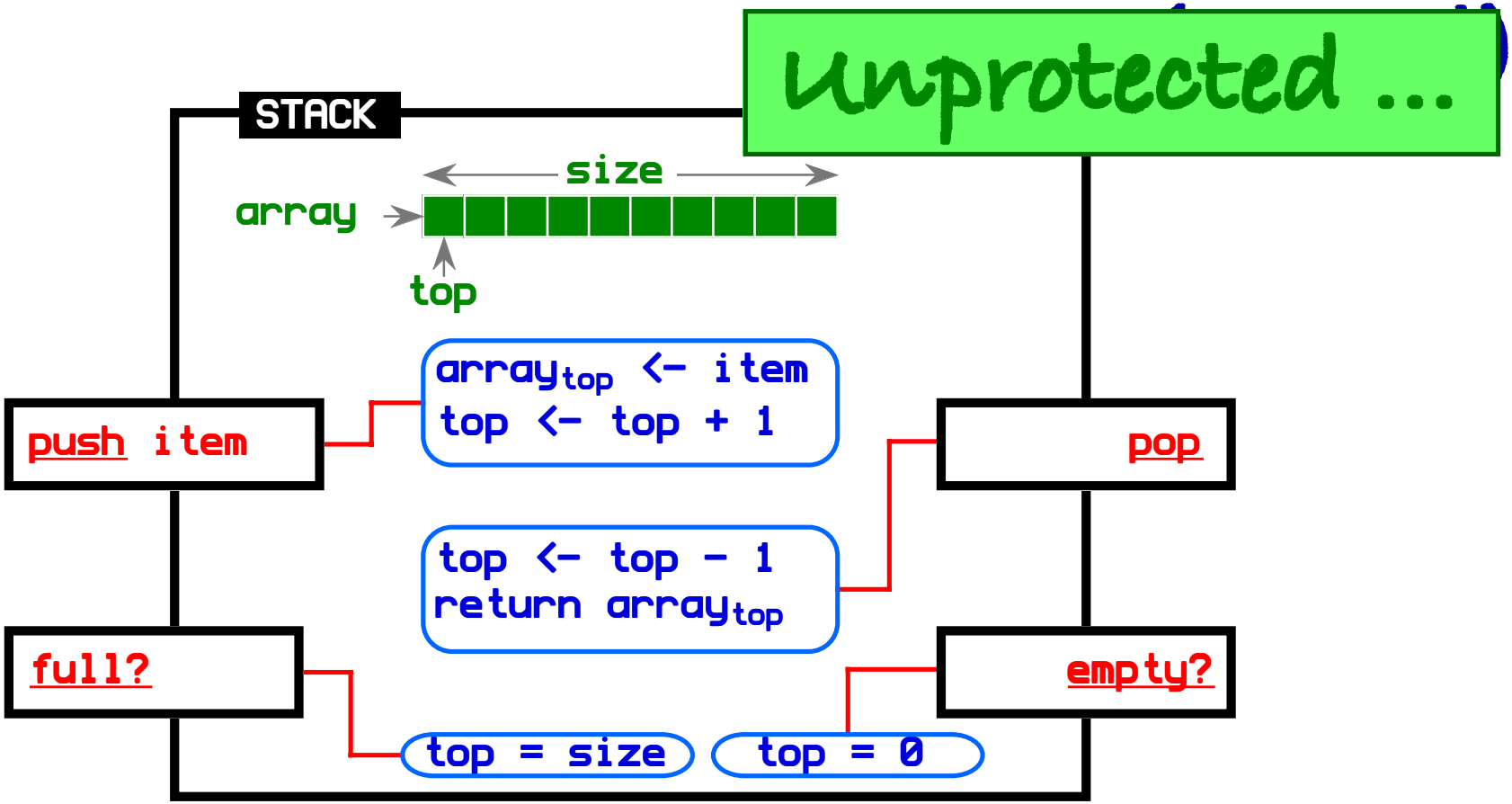
```
length month year
  if month = 2
    if leap-year? year
      29 28
    if even? month = (month > 7)
      31 30
```

```
julian->date day year
  loop day month
   if day > length month year
     loop day - length month year
          month + 1
     list day month year
  loop day 1
```

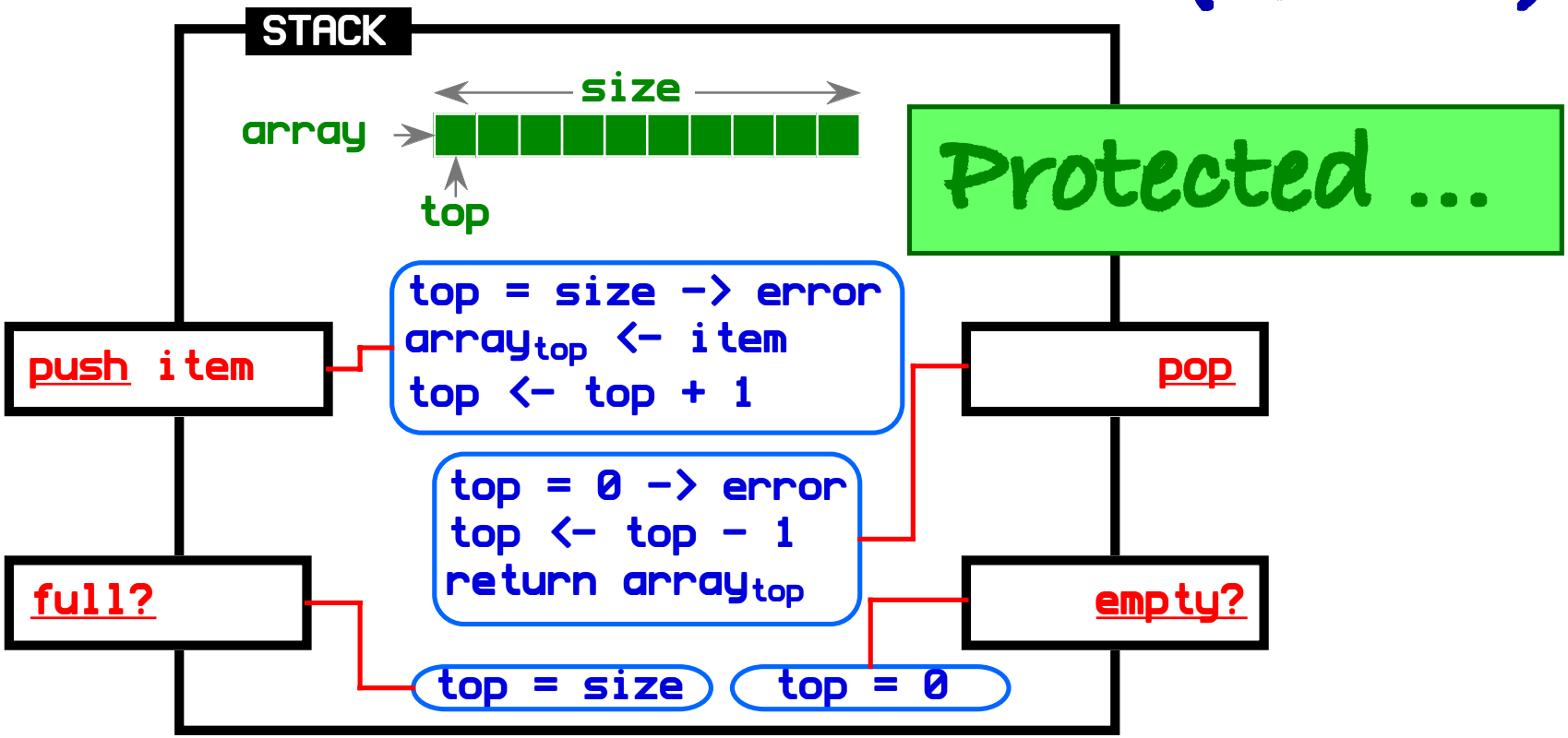
Abstraction: data abstraction

composition of one or more (procedural) abstractions around a shared, persistent and hidden data structure

Abstraction: data abstraction



Abstraction: data abstraction (cont'd)



Abstraction: data abstraction (cont'd)

```
A is a stack
A:push 'alfa'
A:push 'omega'
while not A:empty?
   display A:pop
```

```
B is a protected stack
c is 0
repeat
   B:push c
   c<-c+1
until B:full?
display B:pop
```

Abstraction: data abstraction (cont'd)

```
A is a stack
A:push 'alfa'
A:push 'omega'
while not A:empty?
   display A:pop
```

Same signatures ...

```
B is a protected stack
c is 0
repeat
   B:push c
   c<-c+1
until B:full?
display B:pop
```

Abstraction: data abstraction

```
A is a stack
A:push 'alfa'
A:push 'omega'
while not A:empty?
   display A:pop
```

Different implementations ...

```
B is a protected stack
c is 0
repeat
   B:push c
   c<-c+1
until B:full?
display B:pop
```

Contents (cont'd)

- ☑ Definition
- ☑ Abstraction
- ☐ Polymorphism
  - Definition
  - Structured polymorphism
  - Incremental specification
- ☐ Inheritance
- ☐ ...

Definition of polymorphism

- generic behaviour patterns (i.e. with multiple implementations)
- implementation selection on the basis of object types in context
- formalizes conventions
- promotes substitutability

Structured polymorphism

Object orientation

=

data abstraction

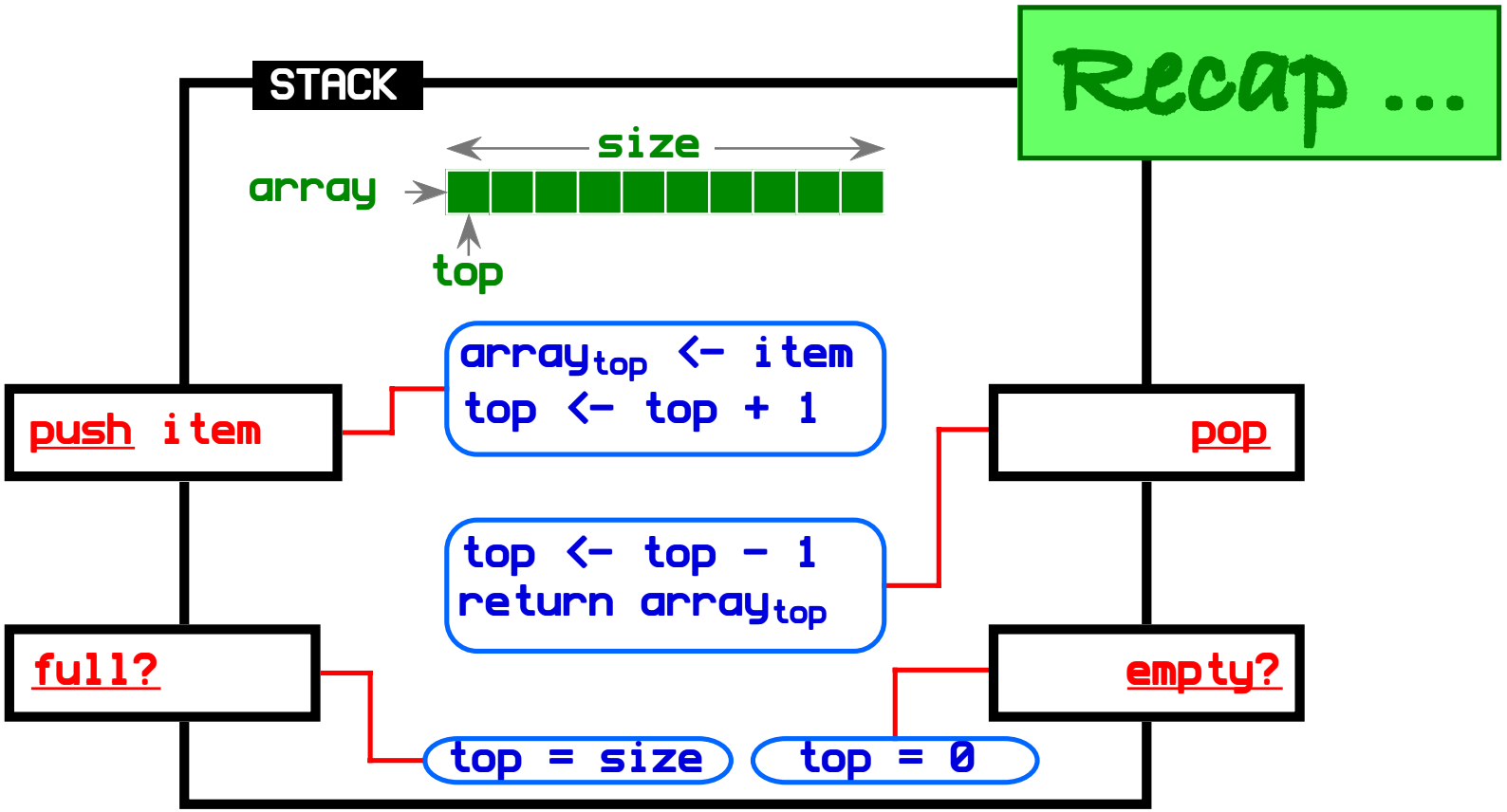
+

structured polymorphism

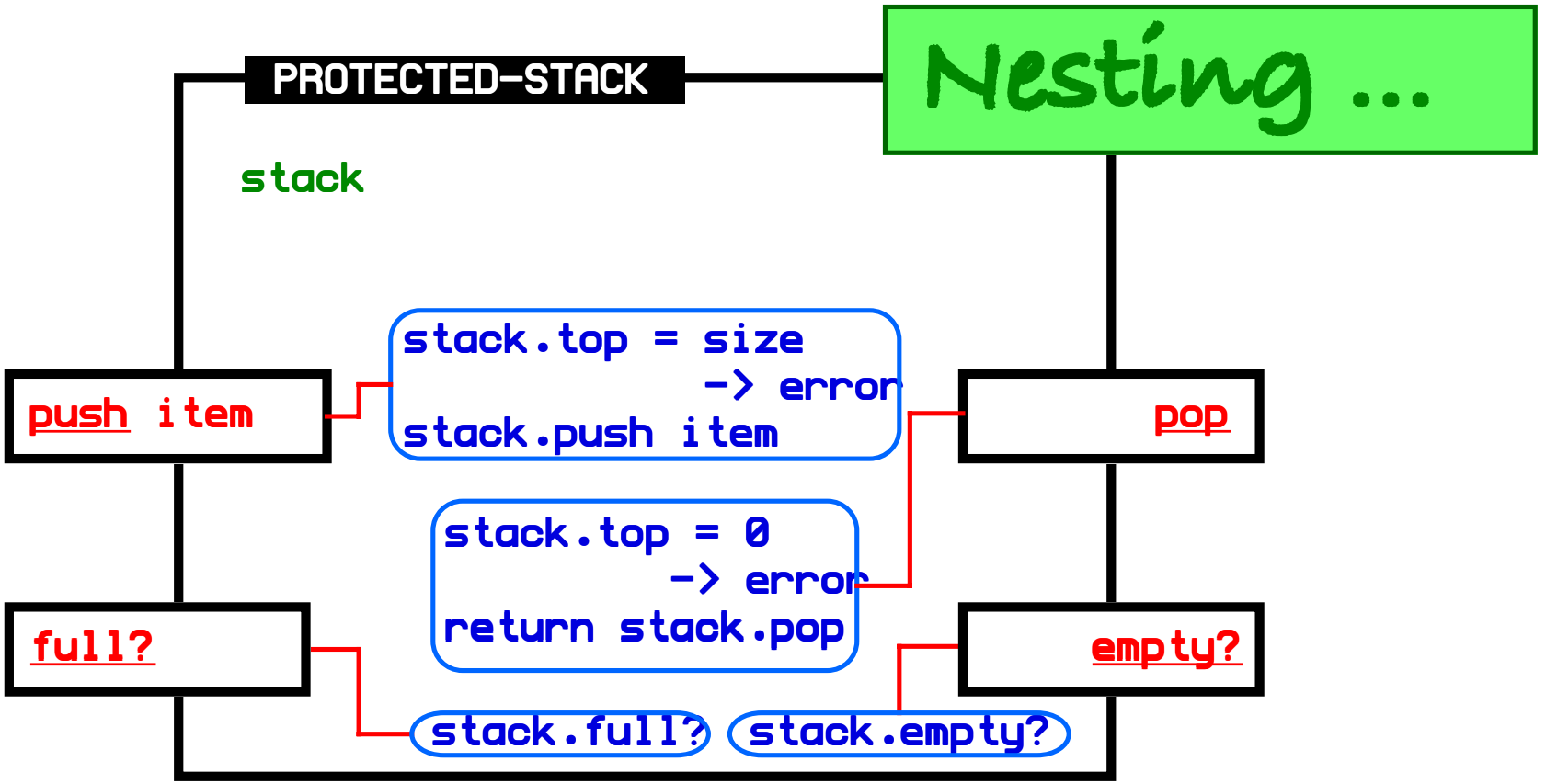
Structured polymorphism (cont'd)

- incremental abstractions
- inheritance of features
- organized genericity
- framework constraints

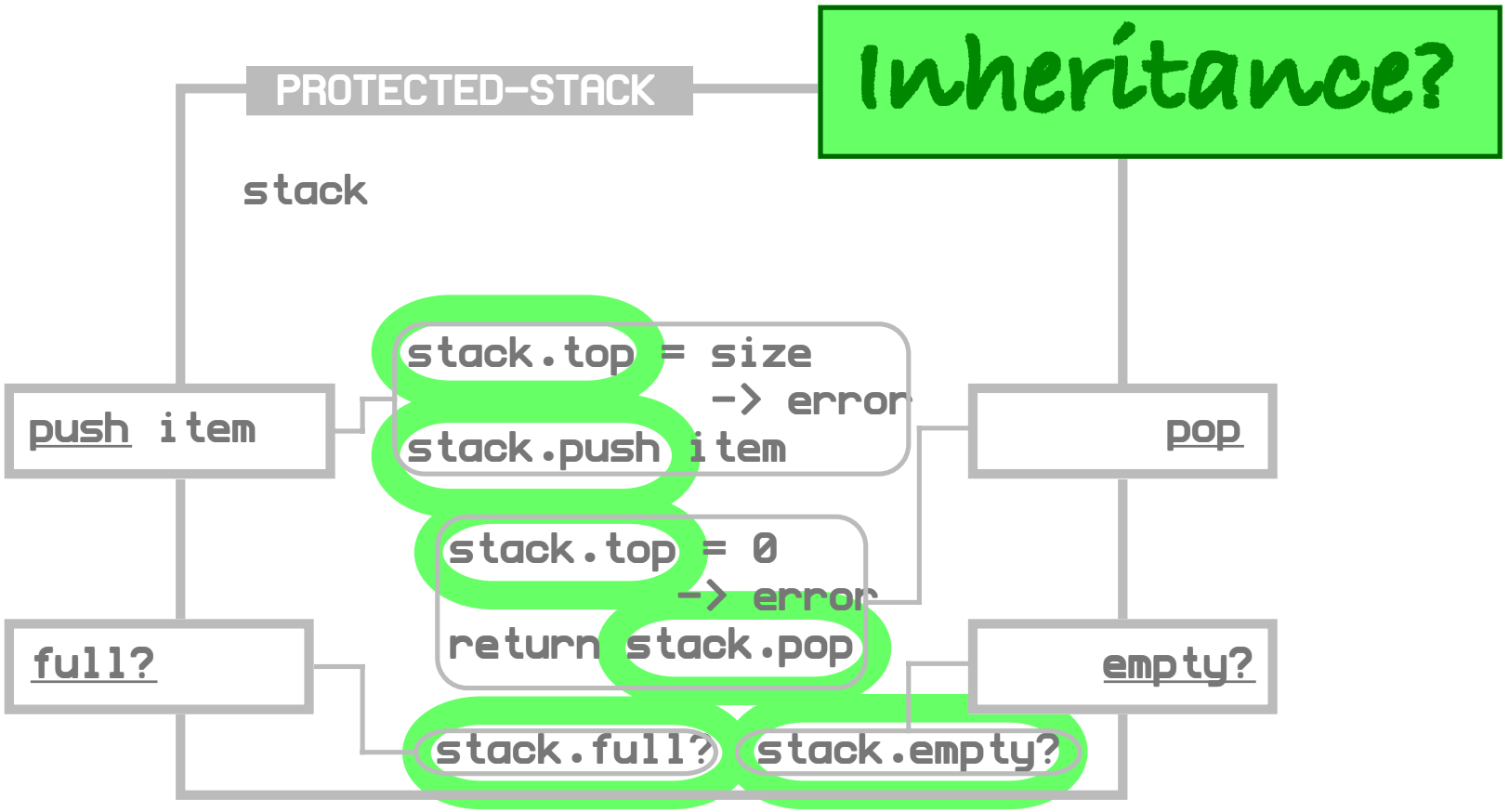
Incremental specification



Incremental specification (cont'd)



Incremental specification (cont'd)



Contents (cont'd)

- ☑ ...
- ☑ Polymorphism
- ☐ Inheritance
  - Class based inheritance
  - Prototype based inheritance
  - Actor based inheritance
- ☐ Frameworks
- ☐ ...

Class based inheritance

μ is a signature...

objects belong to the same class if they share the same behaviour

$$A \equiv B$$

$$\Leftrightarrow$$

$$\{\mu \mid A \text{ responds to } \mu\} = \{\mu \mid B \text{ responds to } \mu\}$$

Footnote: $\equiv$ is an equivalence relation

Class based inheritance (cont'd)

In practice:

A class describes ...

... the composition of the internal state ...

... the specification of the common behaviour ...

... of all of its member objects

Class based inheritance (cont'd)

A taxonomy is defined on the set of all classes, e.g.:

A class is a subclass of another class if it has at least the same behaviour

$$A \approx B$$

$$\Leftrightarrow$$

$$A \text{ specifies } \mu \Longrightarrow B \text{ specifies } \mu$$

Footnote: $\approx$ is a partial order

Class based inheritance (cont'd)

In practice:

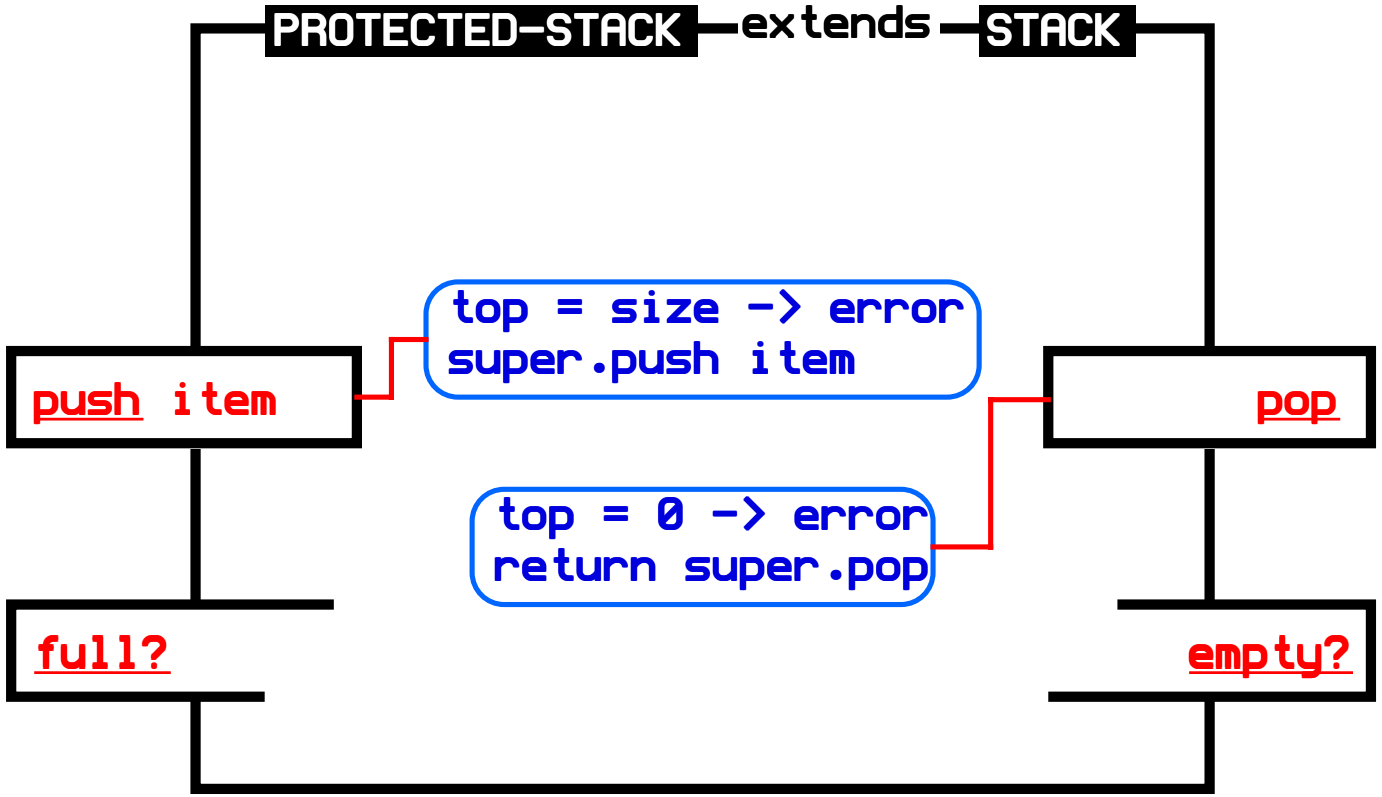
A subclass has ...

... at least the same composition of the internal state ...

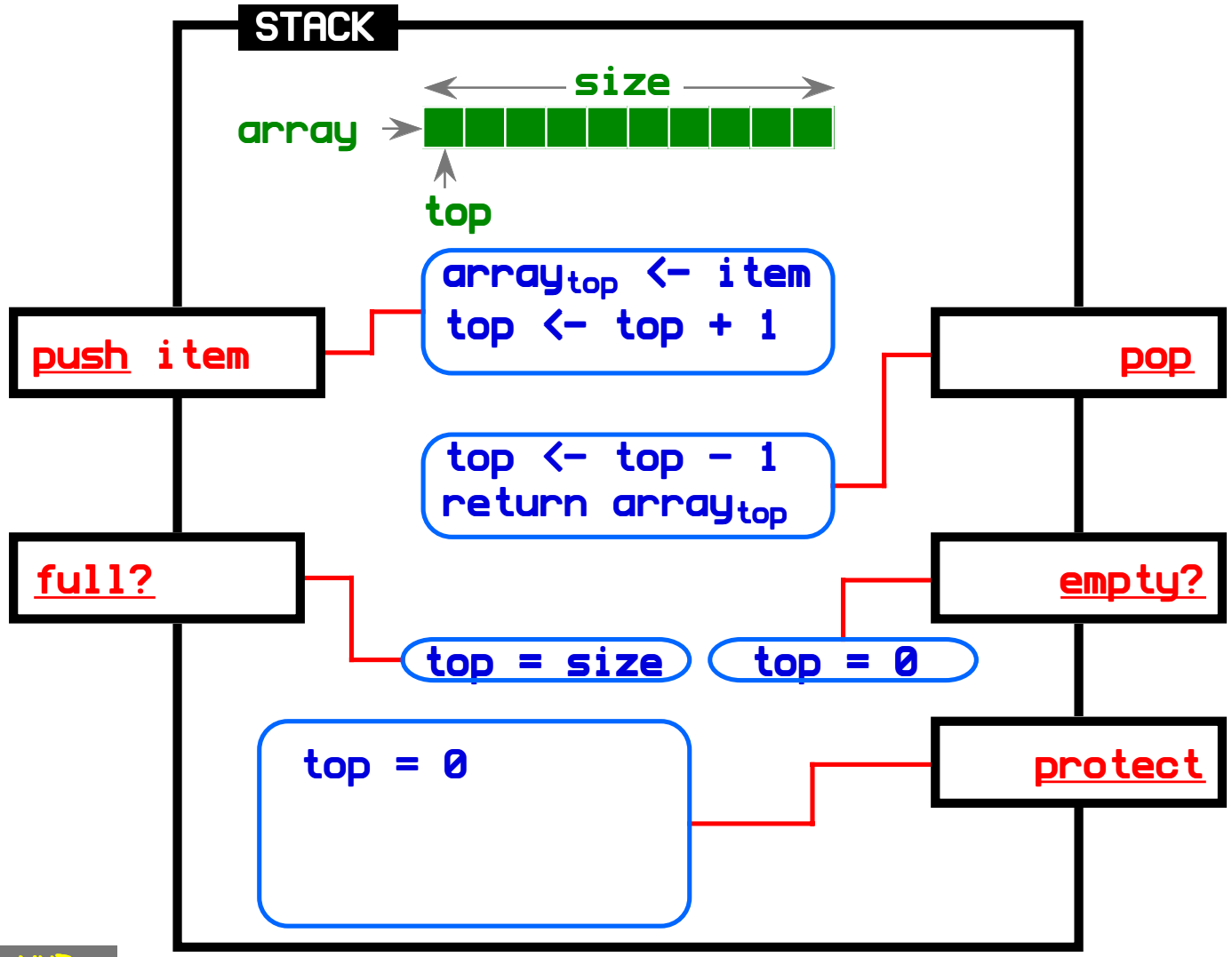
... at least the same specification of behaviour ...

... as its superclass

Class based inheritance (cont'd)



Prototype based inheritance



Prototype based inheritance (cont'd)

Actor based inheritance

Actor based inheritance (cont'd)

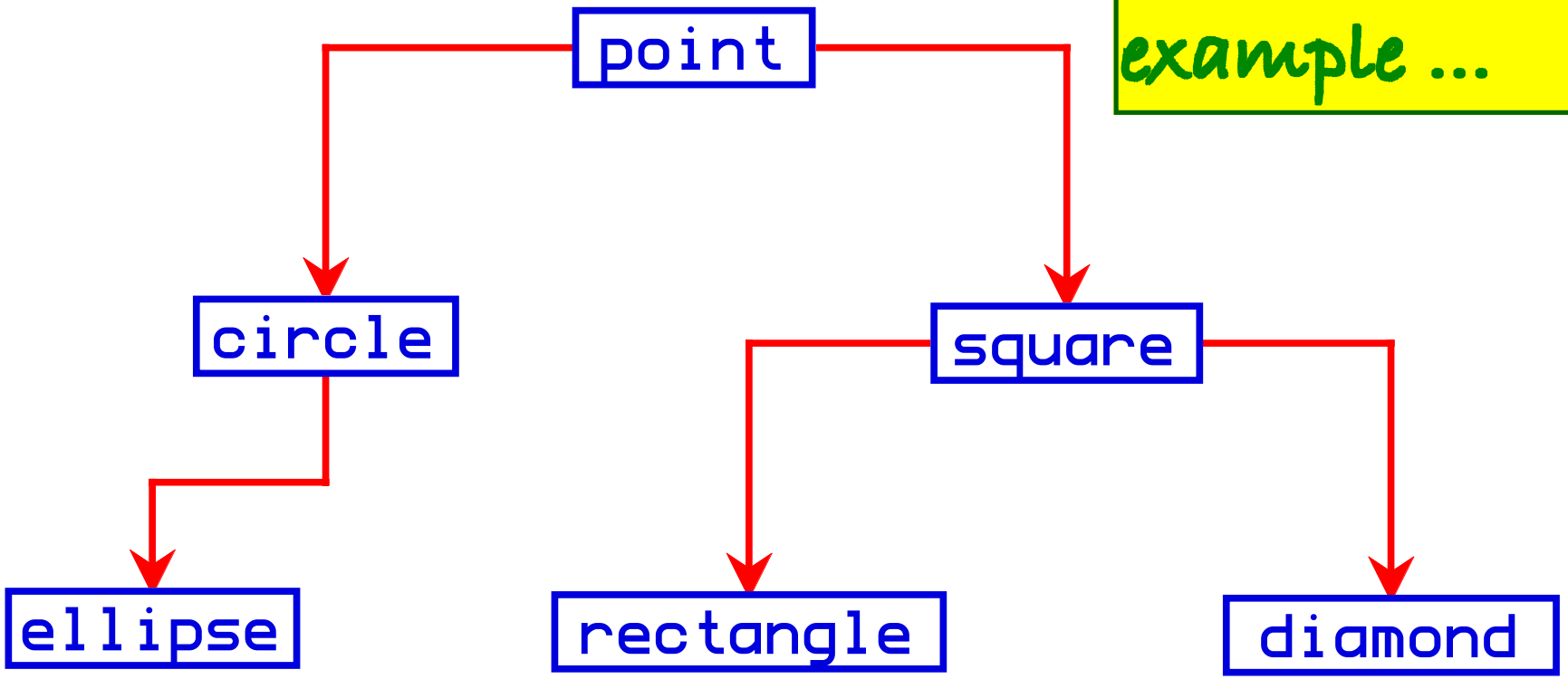
Actor based inheritance (cont'd)

Contents (cont'd)

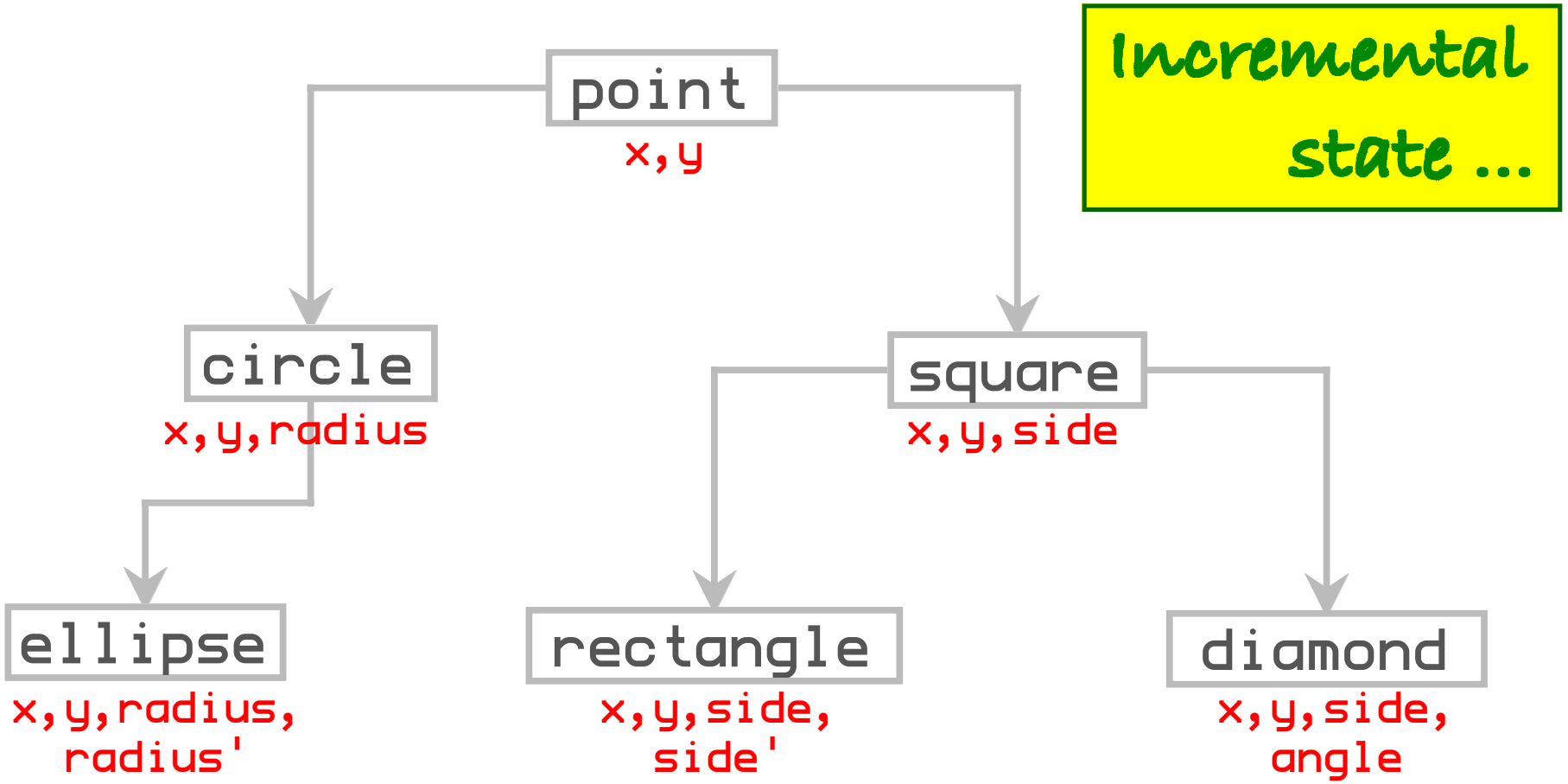
- ☑ ...
- ☑ Inheritance
- ☐ Frameworks
  - Frameworks
  - Overriding
  - Example
  - Problems
- ☐ Simulation
- ☐ ...

Frameworks

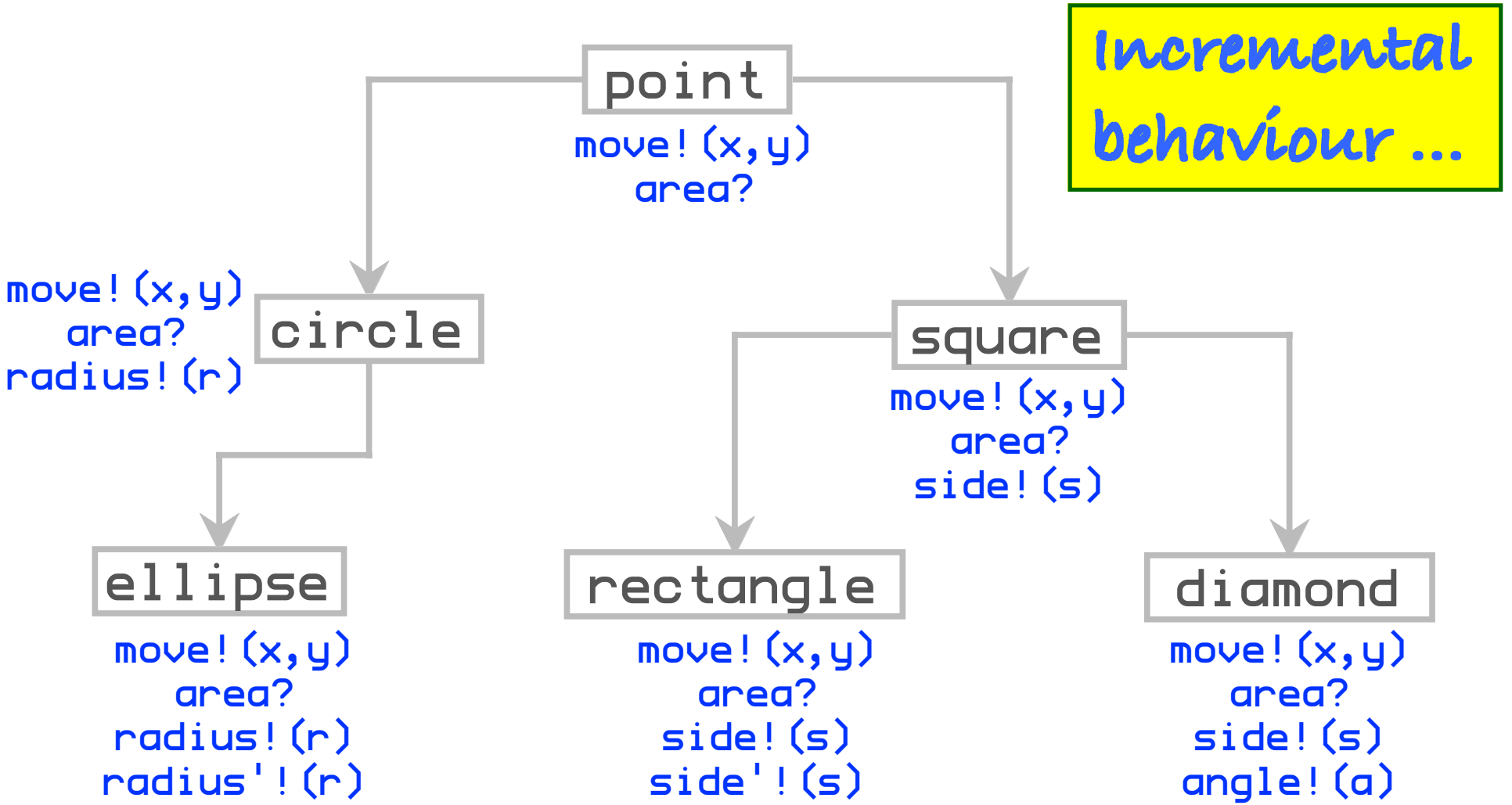
Ubiquitous example ...



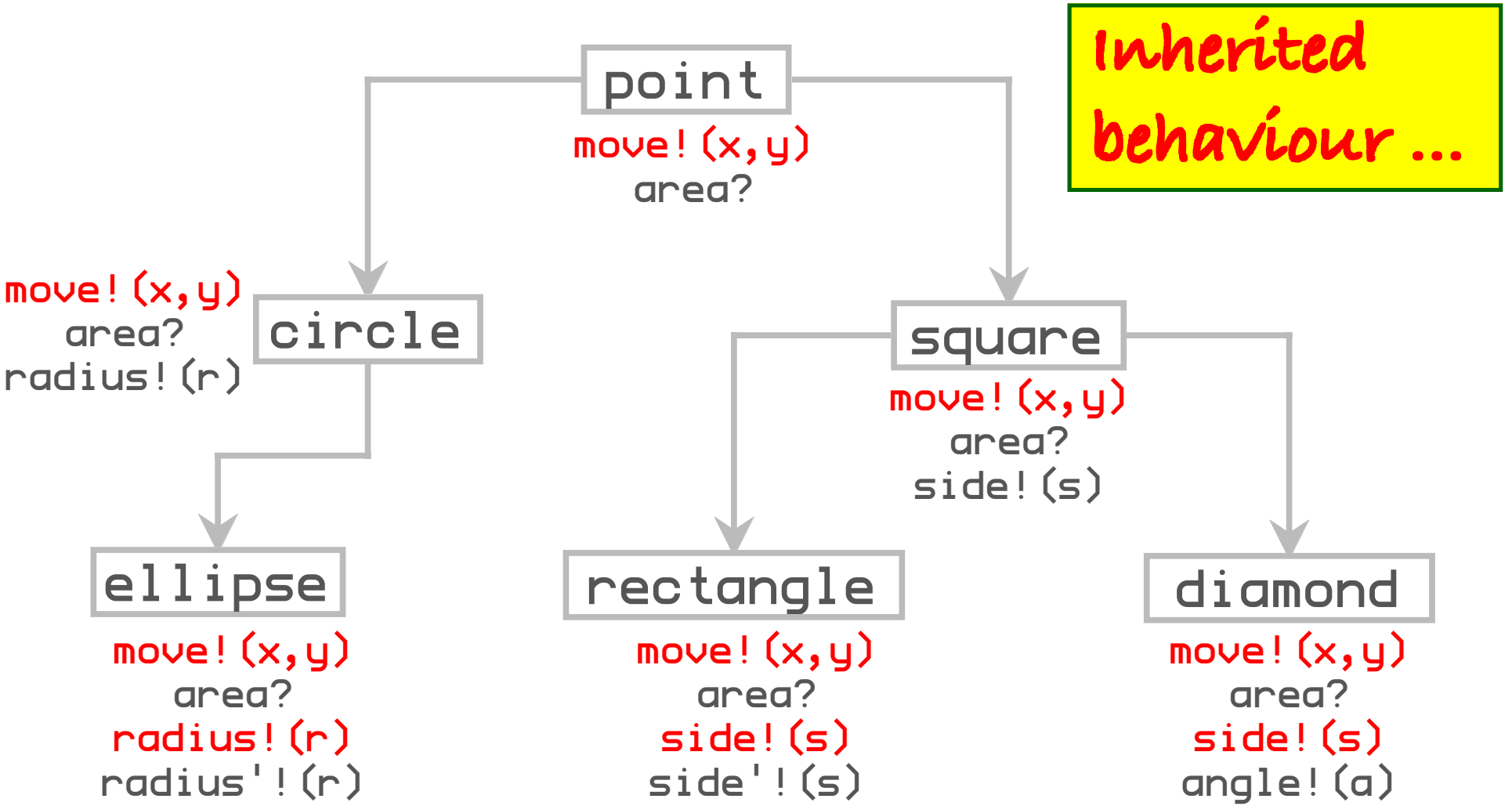
Frameworks (cont'd)



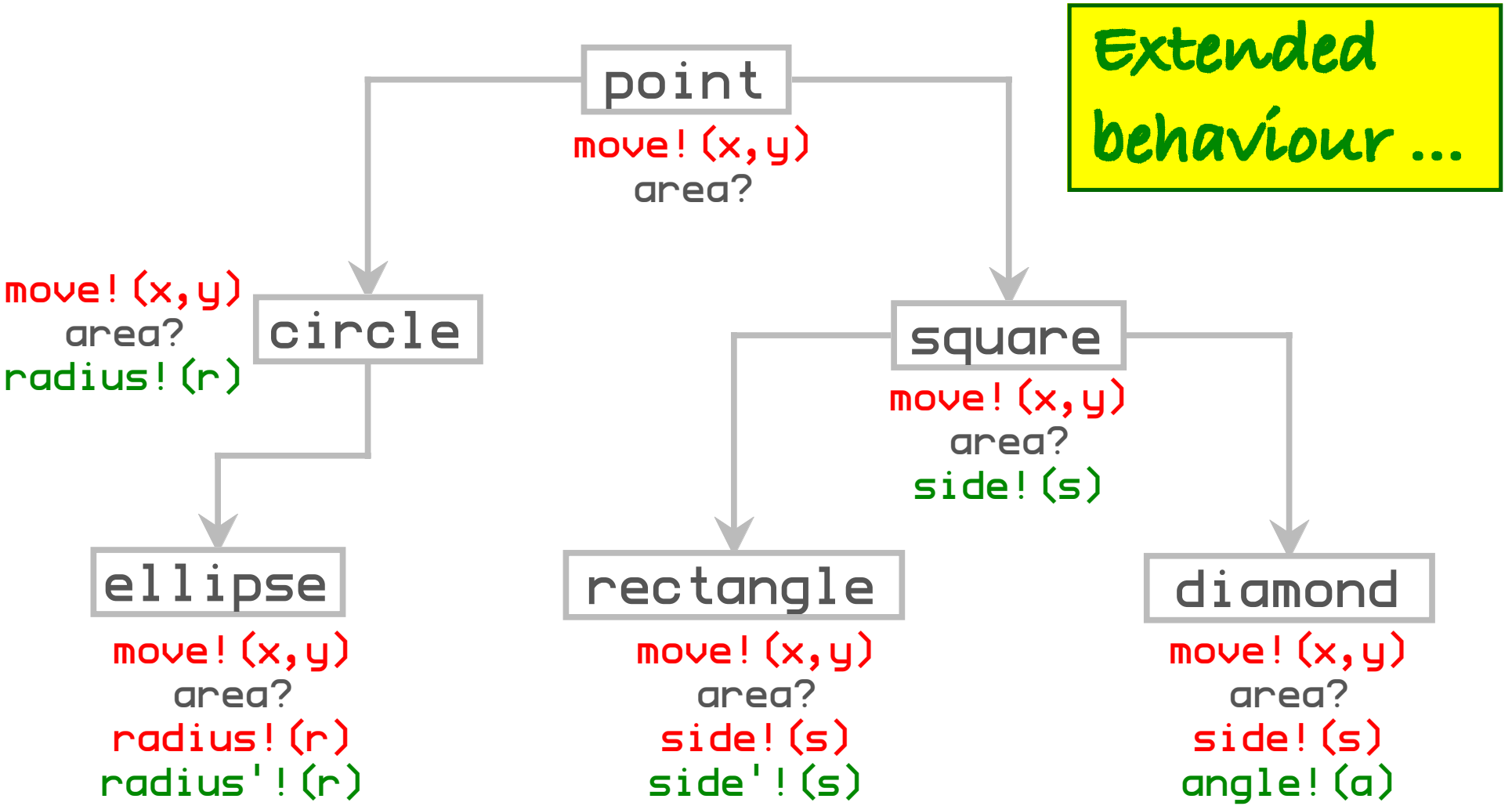
Frameworks (cont'd)



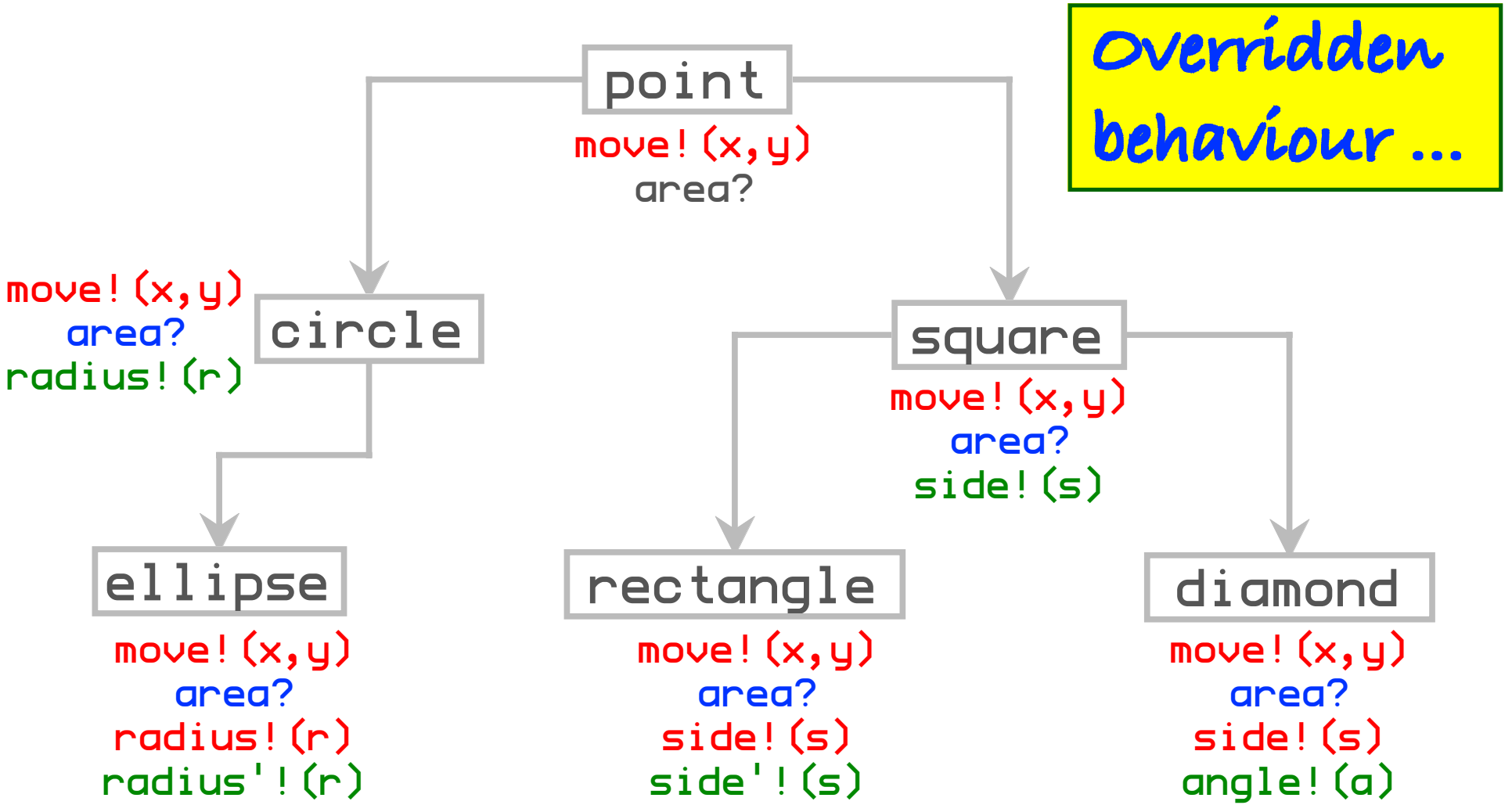
Frameworks (cont'd)



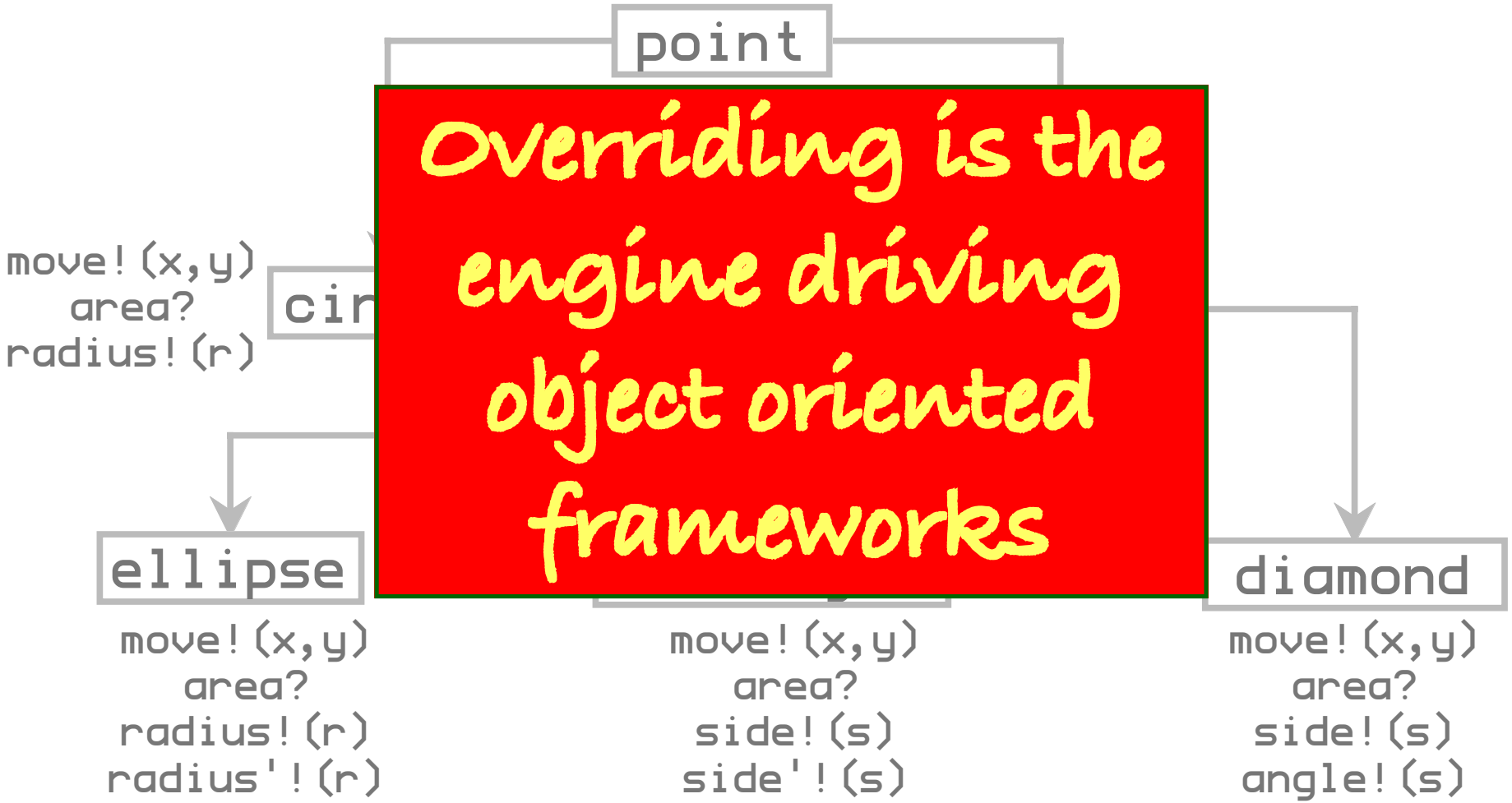
Frameworks (cont'd)



Frameworks (cont'd)



Frameworks: overriding



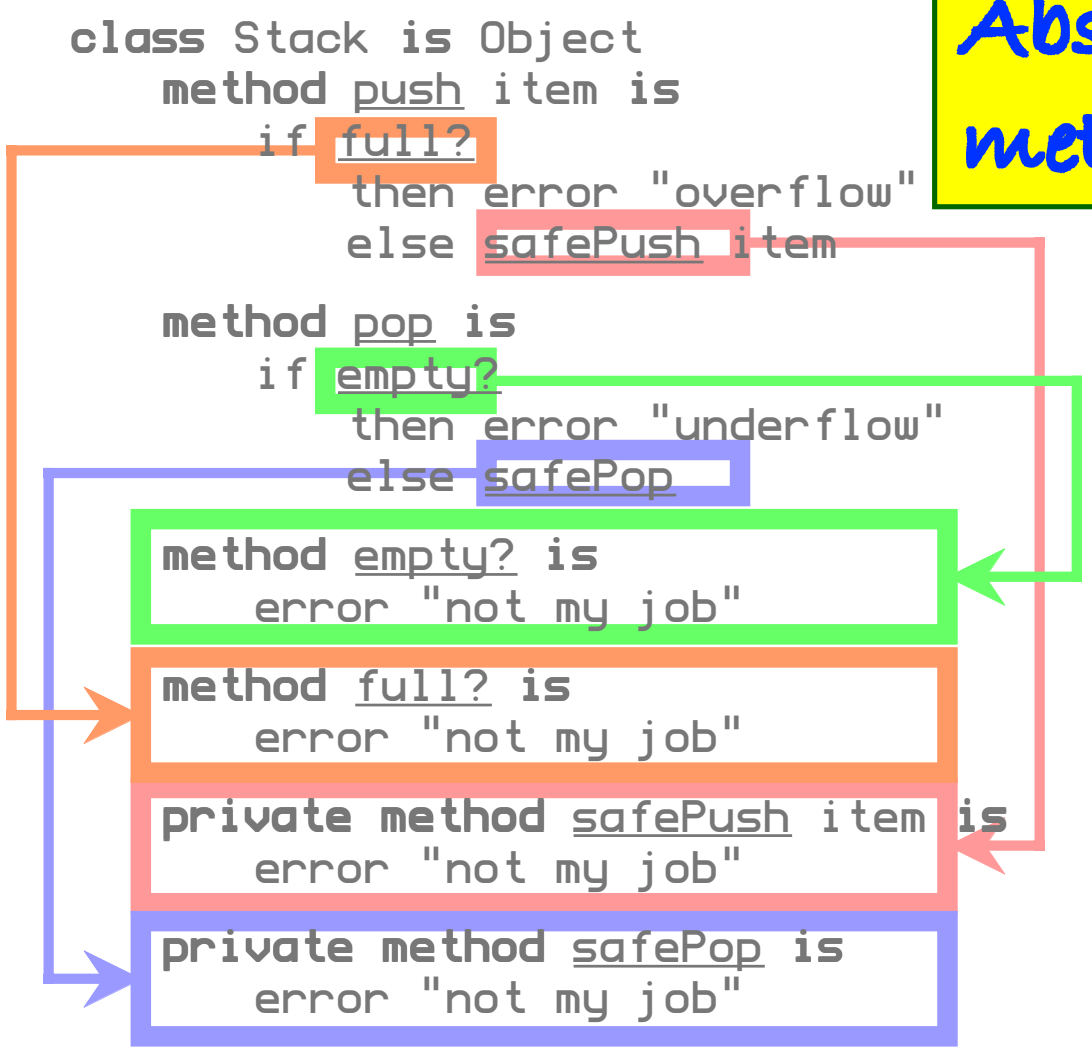
Overriding is the engine driving object oriented frameworks

Framework example

```
class Stack is Object
    method push item is
        if full?
            then error "overflow"
            else safePush item

    method pop is
        if empty?
            then error "underflow"
            else safePop

    method empty? is
        error "not my job"

    method full? is
        error "not my job"

    private method safePush item is
        error "not my job"

    private method safePop is
        error "not my job"
```

Framework example (cont'd)

Abstract methods ...



Framework example (cont'd)

```
class ArrayStack is Stack
    variable top is 0
    variable size is 10
    variable array is Array size 10

    method empty? is
        top equal 0

    method full? is
        top greater size

    private method safePush item is
        array set top item
        top increment

    private method safePop is
        top decrement
        array get top
```

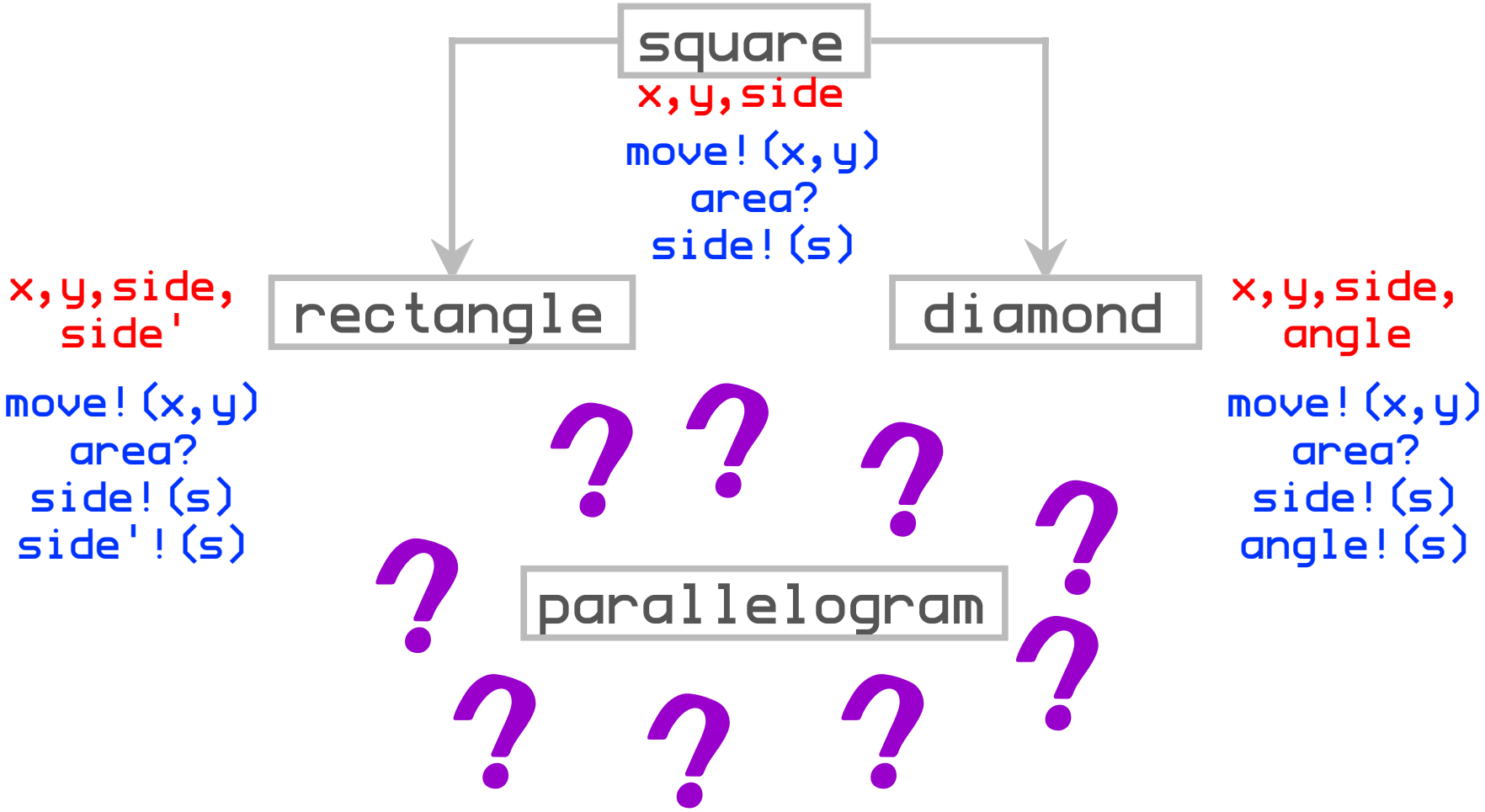
As an array ...

Framework example (cont'd)

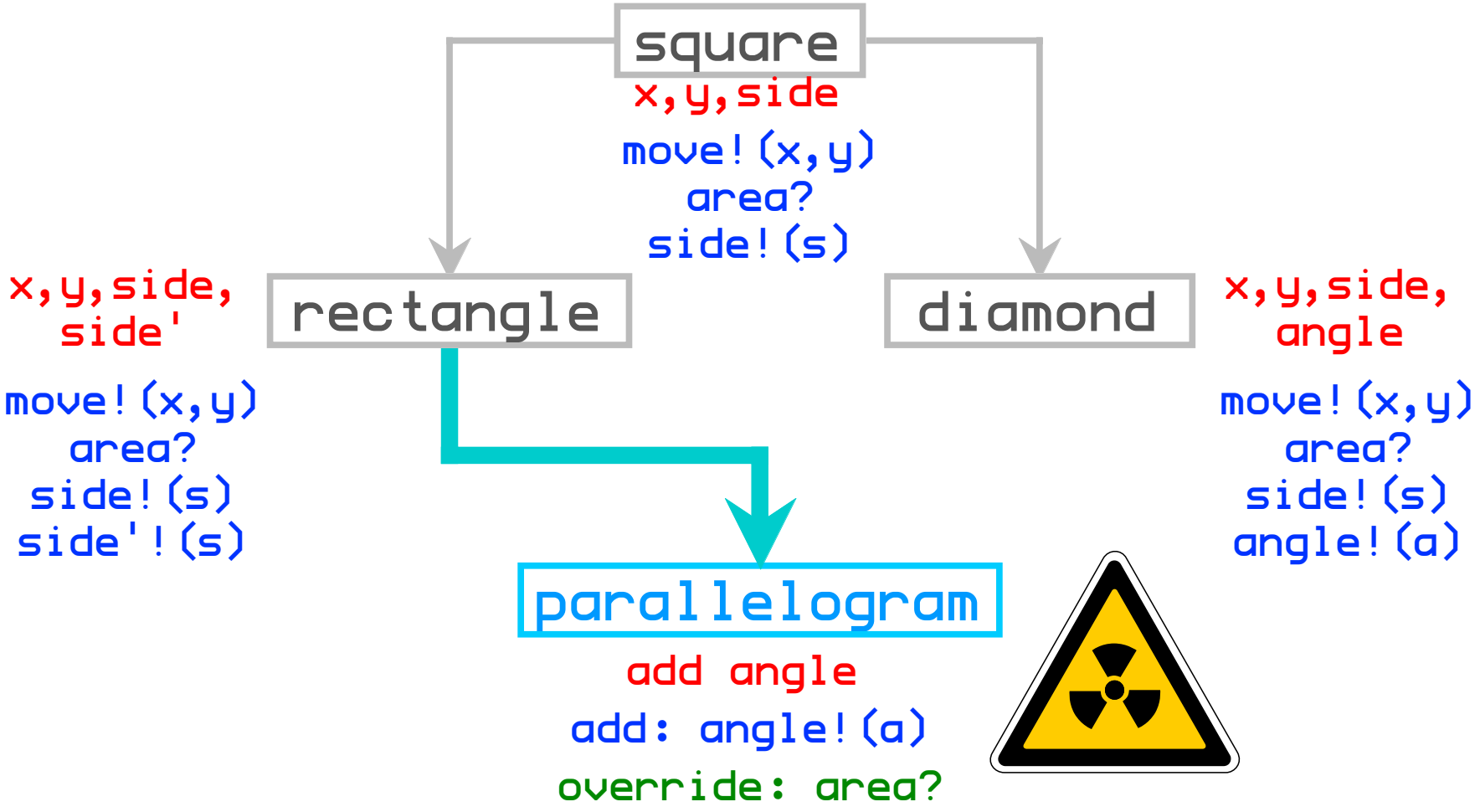
```
class ListStack is Stack
    variable this is null
    variable next is null

    method empty? is
        this equal null

    method full? is
        false

    private method safePush item is
        if this notEqual null
            then next safePush this
        this becomes item

    private method safePop is
        variable hold = this
        this becomes next safePop
        hold
```

As a list ...

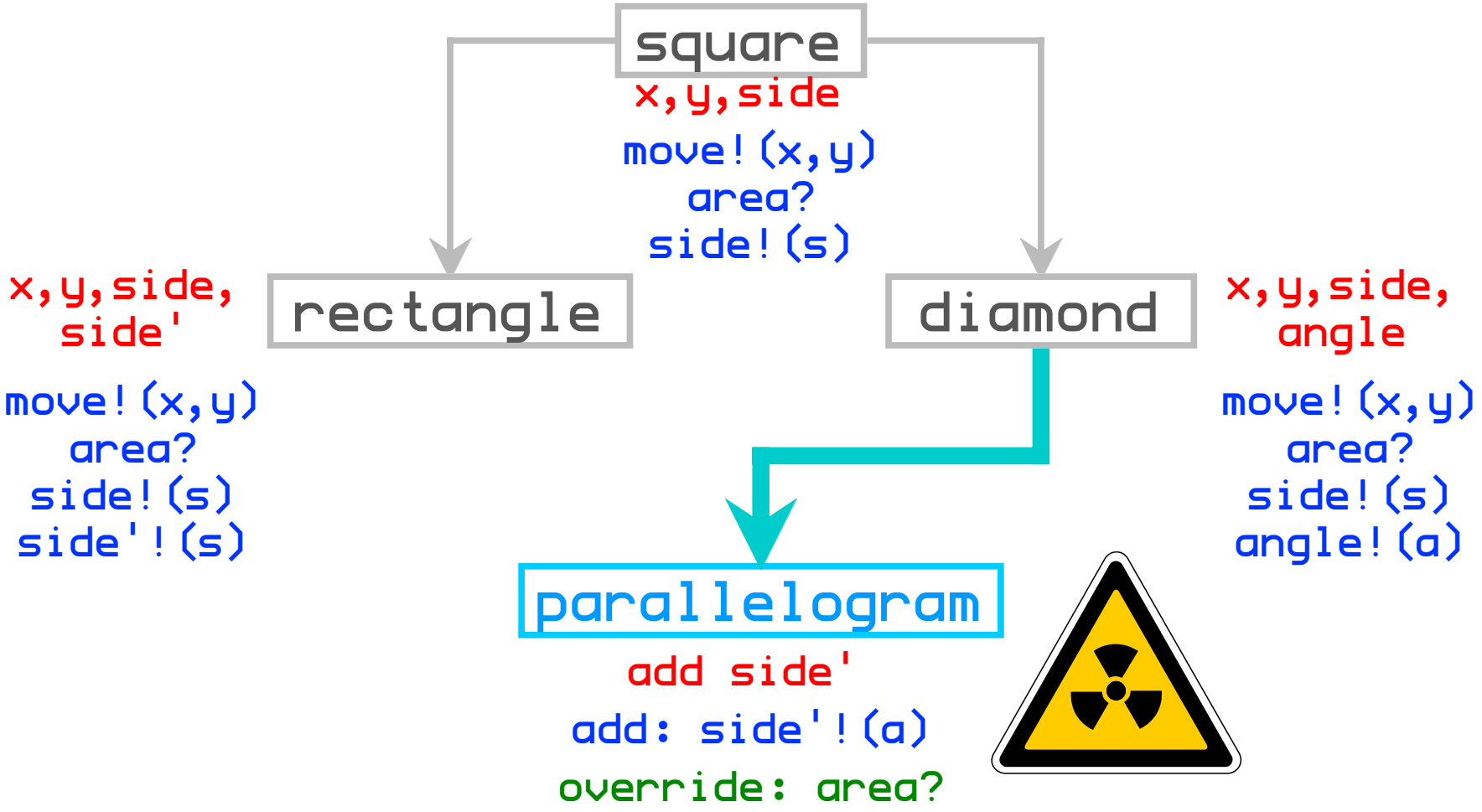
Framework problems



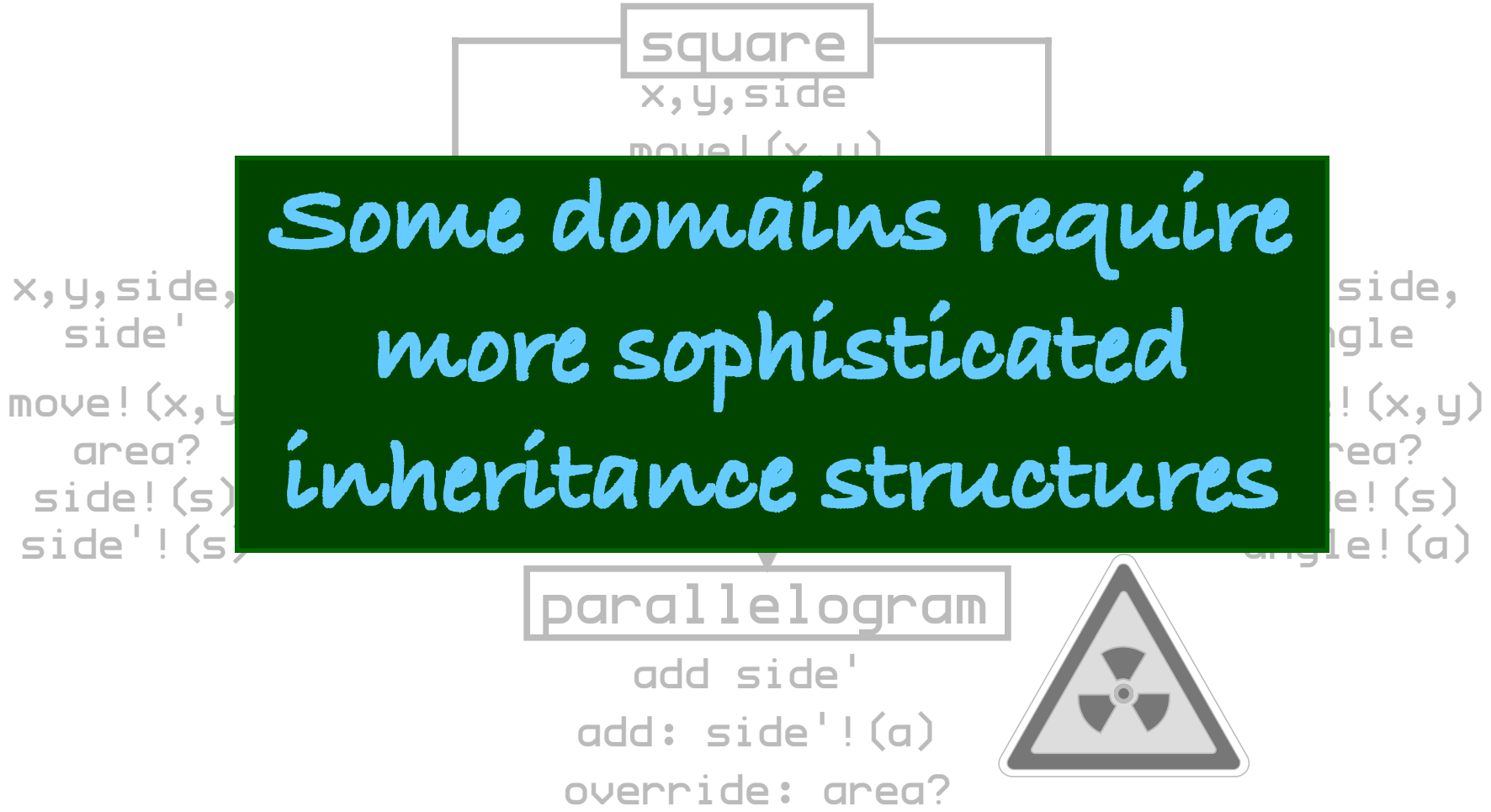
Framework problems (cont'd)



Framework problems (cont'd)

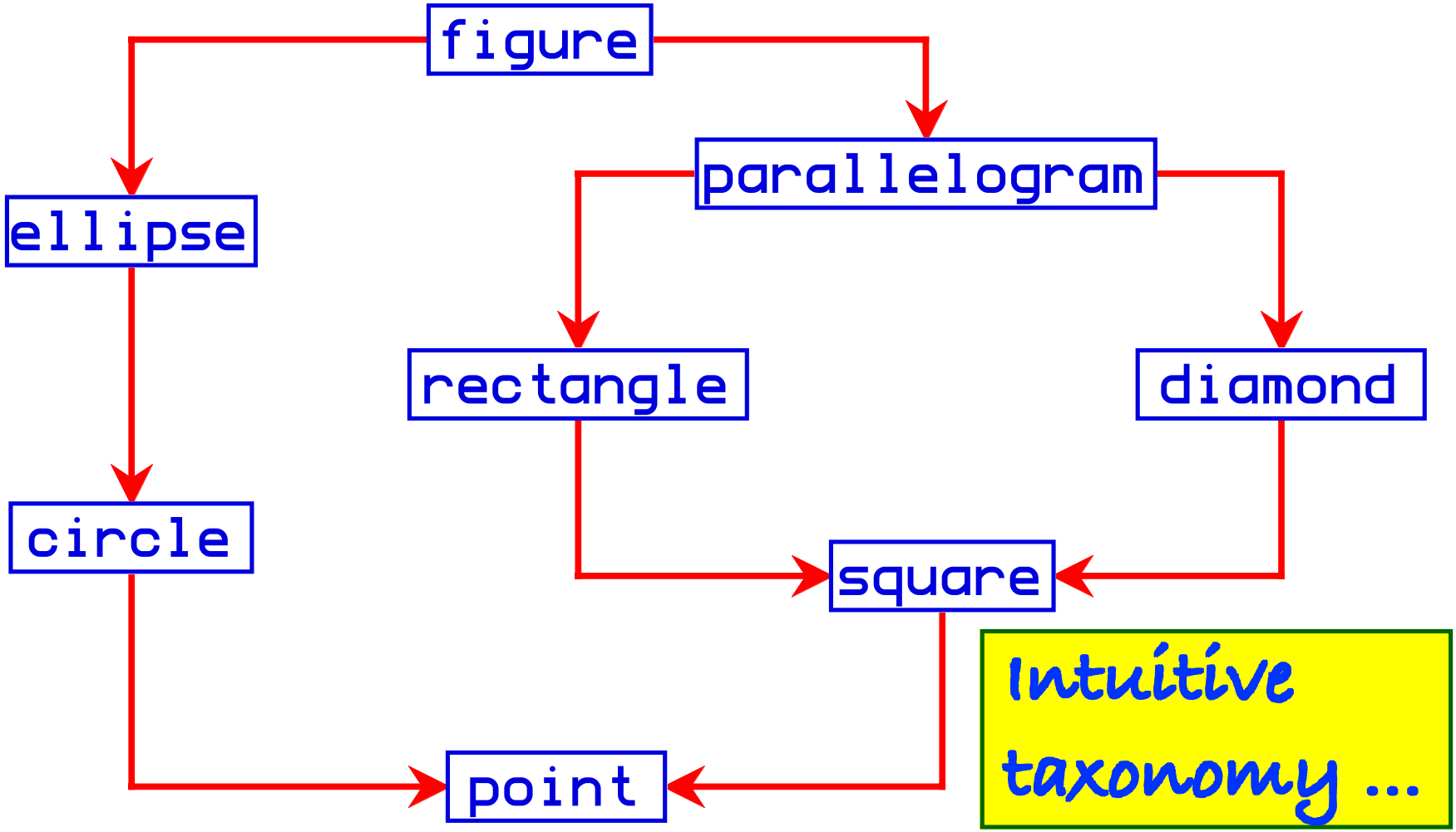


Framework problems (cont'd)

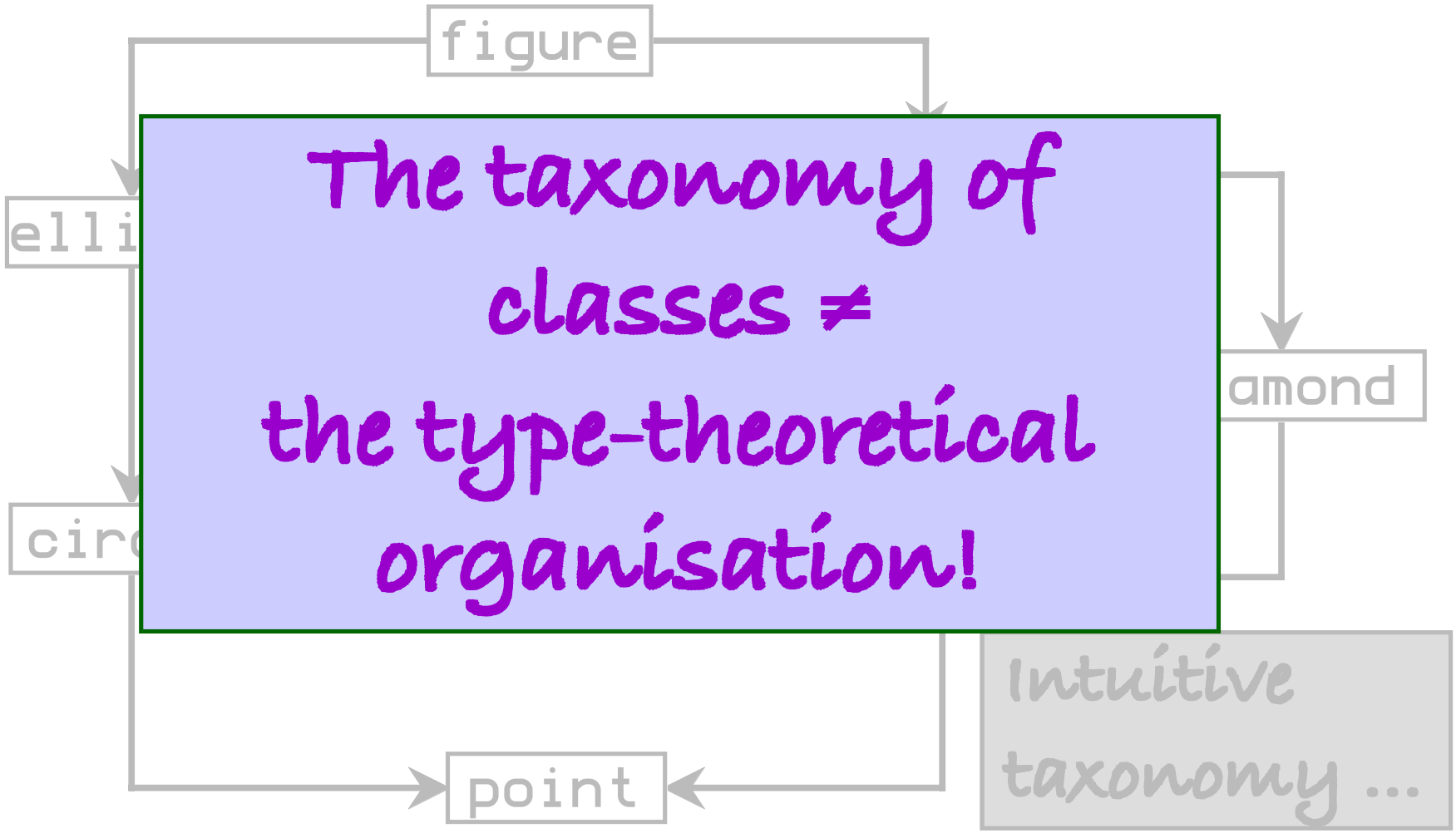


Some domains require more sophisticated inheritance structures

Framework problems (cont'd)



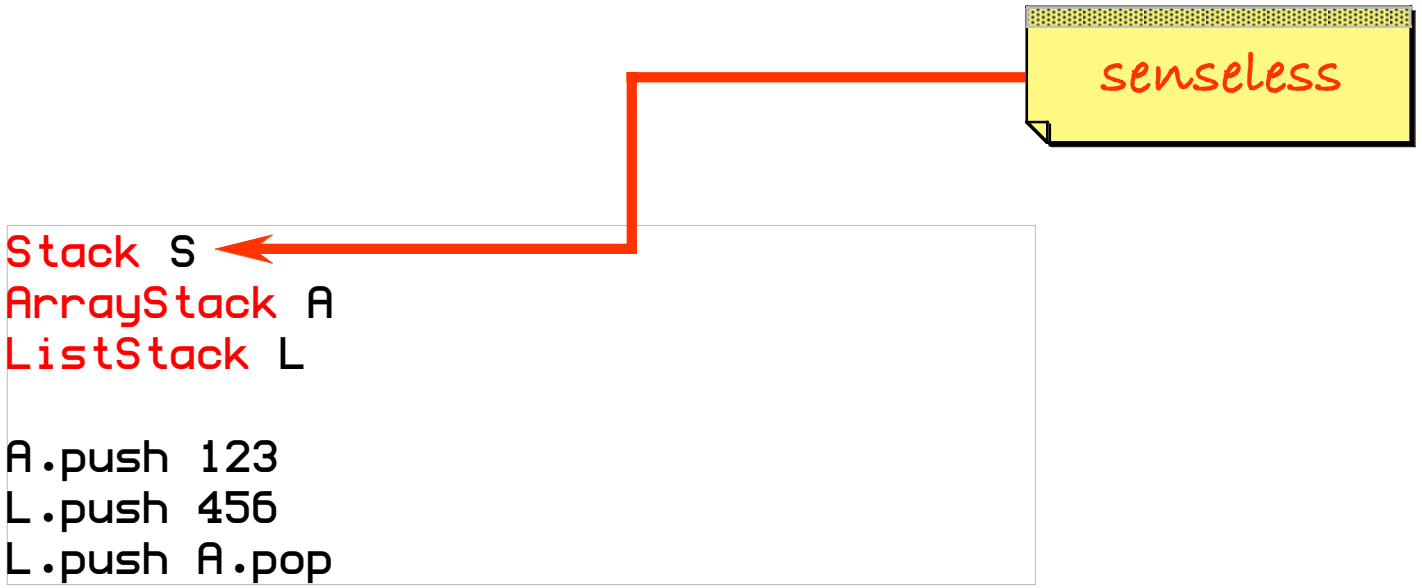
Framework problems (cont'd)



Contents (cont'd)

- ☑ ...
- ☑ Frameworks
- ☐ Types
  - Static types
  - Substitutability
  - Virtual tables
  - Variance
- ☐ Conduits
- ☐ ...

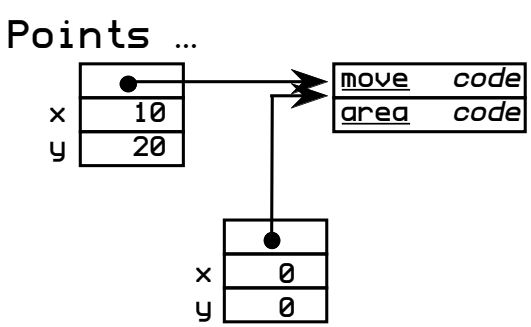
Static types

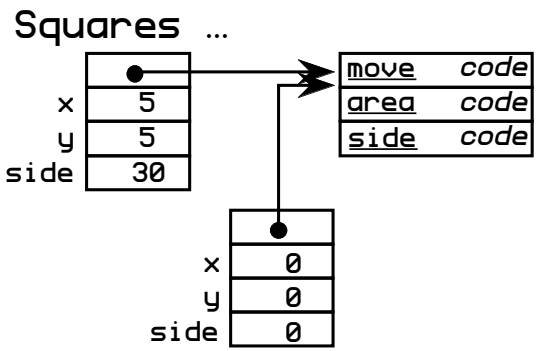


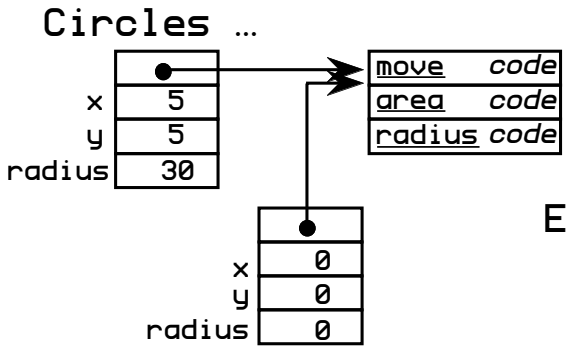
Types: substitutability

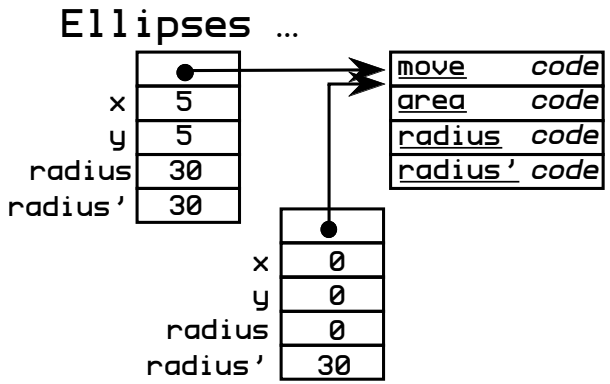
```
Stack S
ArrayStack A
ListStack L

A.push 456
S becomes A
L.push S.pop + 456
S becomes L
S.pop
```

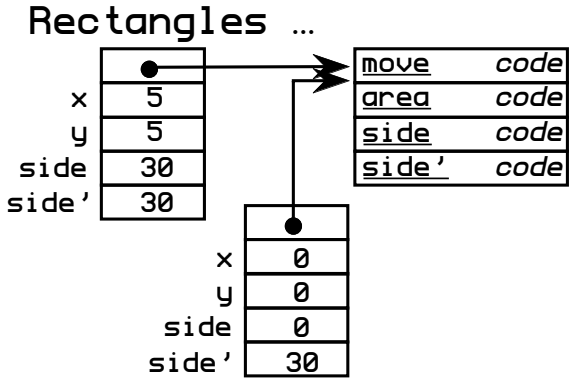
Types: virtual tables











Types: variance

```
class Square
...
method contains Square S
    ...

class Rectangle is Square
...
method contains Rectangle R
    ...
```

overriding?

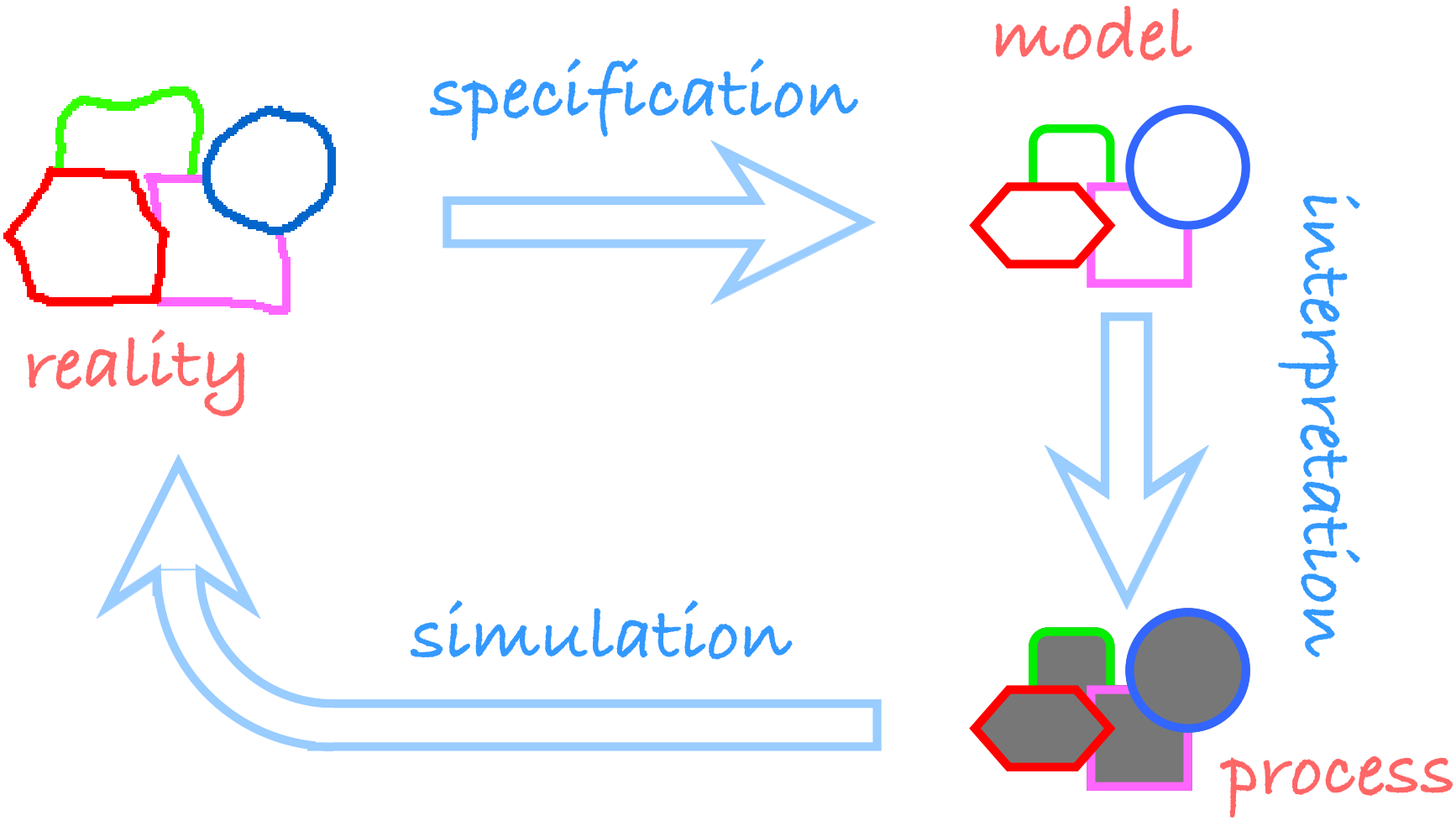
No, because contains is going to use side' on a Square!

```
Square S
Rectangle R
...
S becomes R
...
S contains S
```

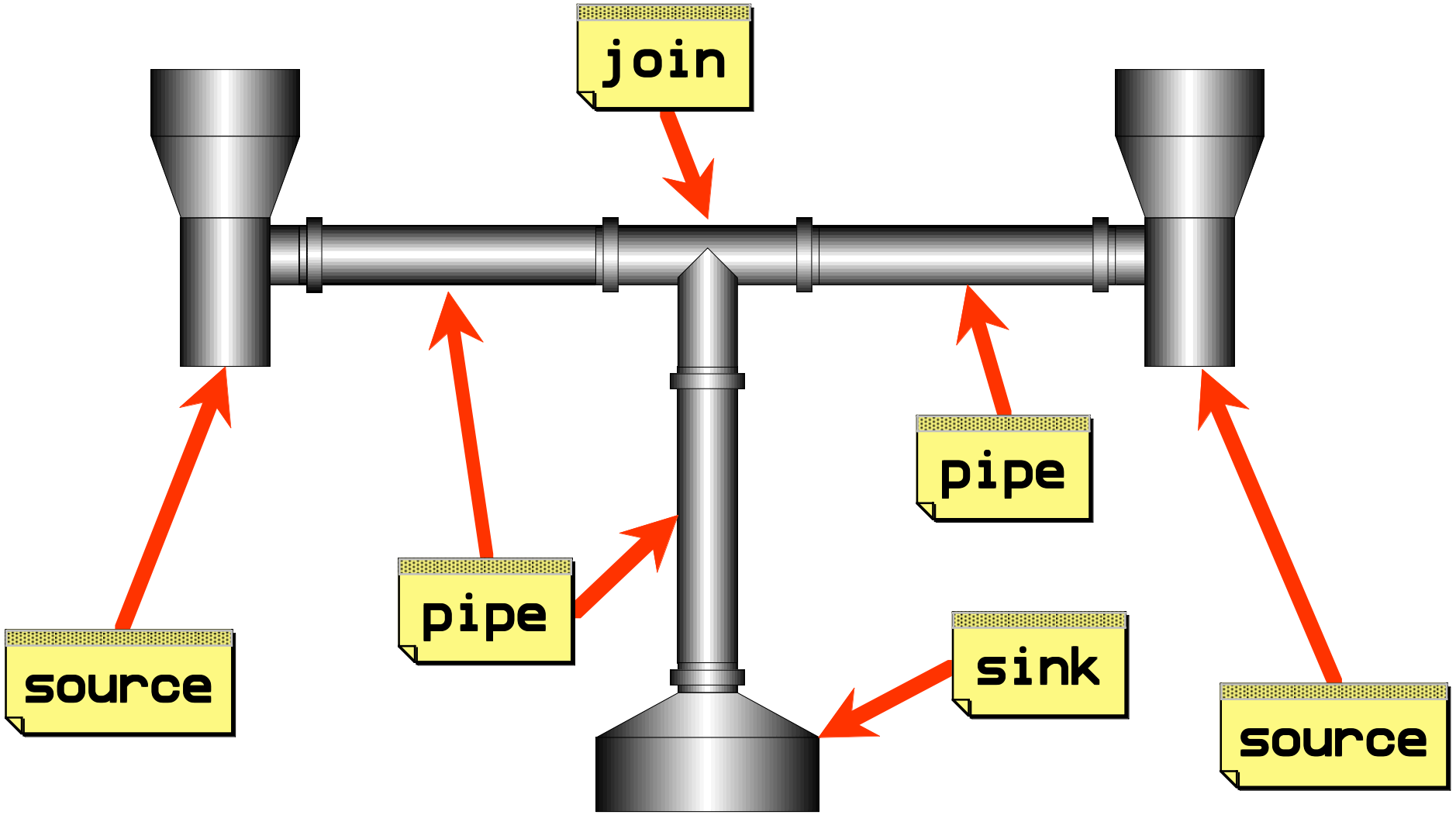
Contents (cont'd)

- ☑ ...
- ☑ Frameworks
- ☑ Types
- ☐ Conduits
  - Simulation
  - Components
  - Specification
  - Implementation

Conduits: simulation



Conduits: simulation (cont'd)

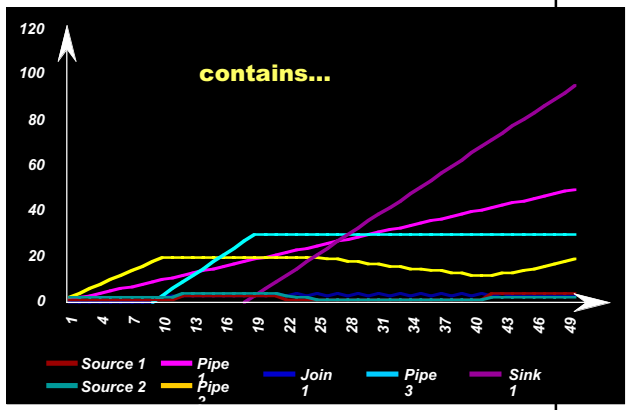


Conduits: simulation (cont'd)

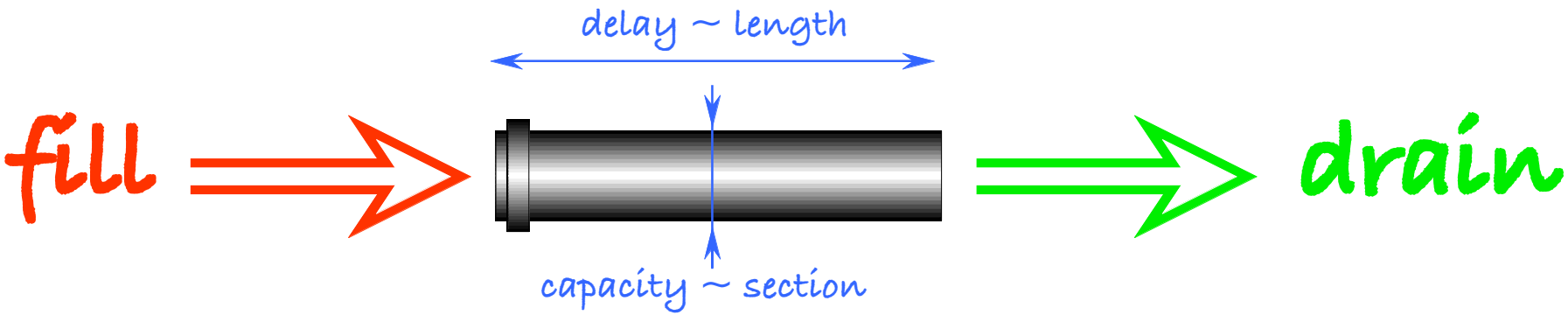
```
PRO_1 is [[10, 1],[20, 3],[40, 1],[50, 4],[80, 1],[90, 5]]
PRO_2 is [[10, 2],[20, 4],[40, 1],[50, 2],[80, 1],[90, 4]]

SRC_1 is Source make "Source 1" 5 PRO_1
PIP_1 is Pipe   make "Pipe   1" 4 SRC_1
SRC_2 is Source make "Source 2" 4 PRO_2
PIP_2 is Pipe   make "Pipe   2" 2 SRC_2
JOI_1 is Join   make "Join   1" 4 PIP_1 PIP_2
PIP_3 is Pipe   make "Pipe   3" 3 JOI_1
SNK_1 is Sink   make "Sink   1" 4 PIP_3
```

Conduits: simulation (cont'd)

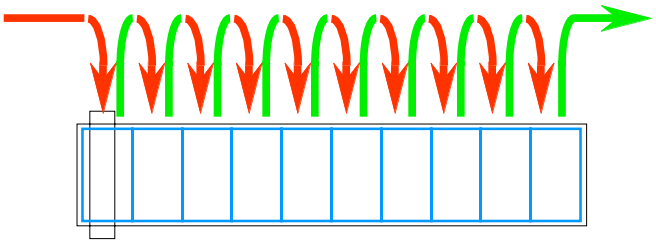
```
t=   0   Source 1   produced    1
t=   1   Source 1   filled      1
t=   1   Source 1   contains    1
t=   1   Source 1   drained     1
t=   1   Pipe   1   filled      1
t=   1   Pipe   1   contains    1
t=   1   Source 1   produced    2
t=   2   Source 1   filled      1
t=   2   Source 1   contains    1
t=   0   Source 2   produced    2
t=   1   Source 2   filled      2
t=   1   Source 2   contains    2
t=   1   Source 2   drained     2
t=   1   Pipe   2   filled      2
t=   1   Pipe   2   contains    2
t=   1   Source 2   produced    4
t=   2   Source 2   filled      2
t=   2   Source 2   contains    2
t=   1   Pipe   1   drained     0
t=   1   Pipe   2   drained     0
```



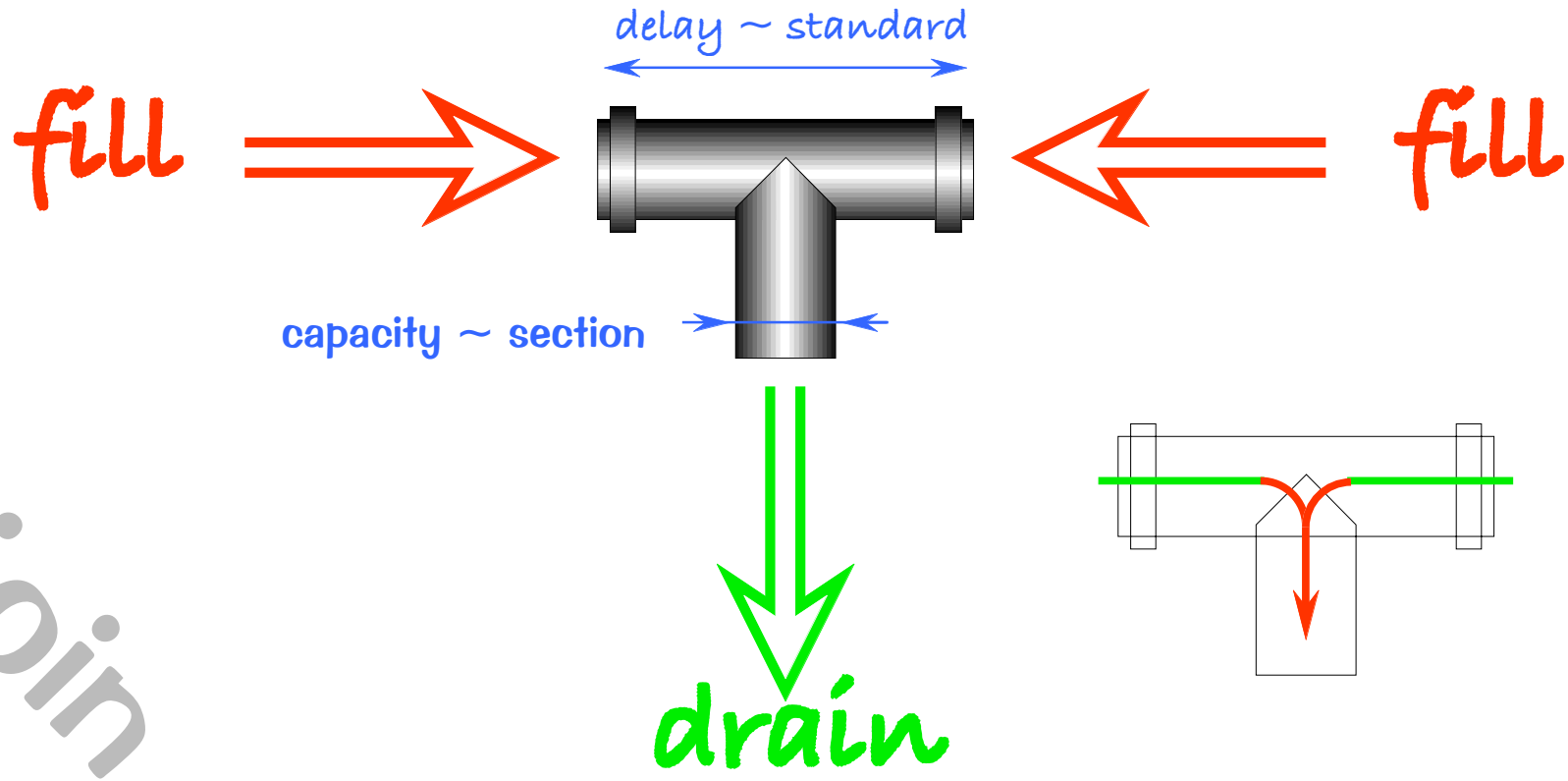
Conduits: components (cont'd)



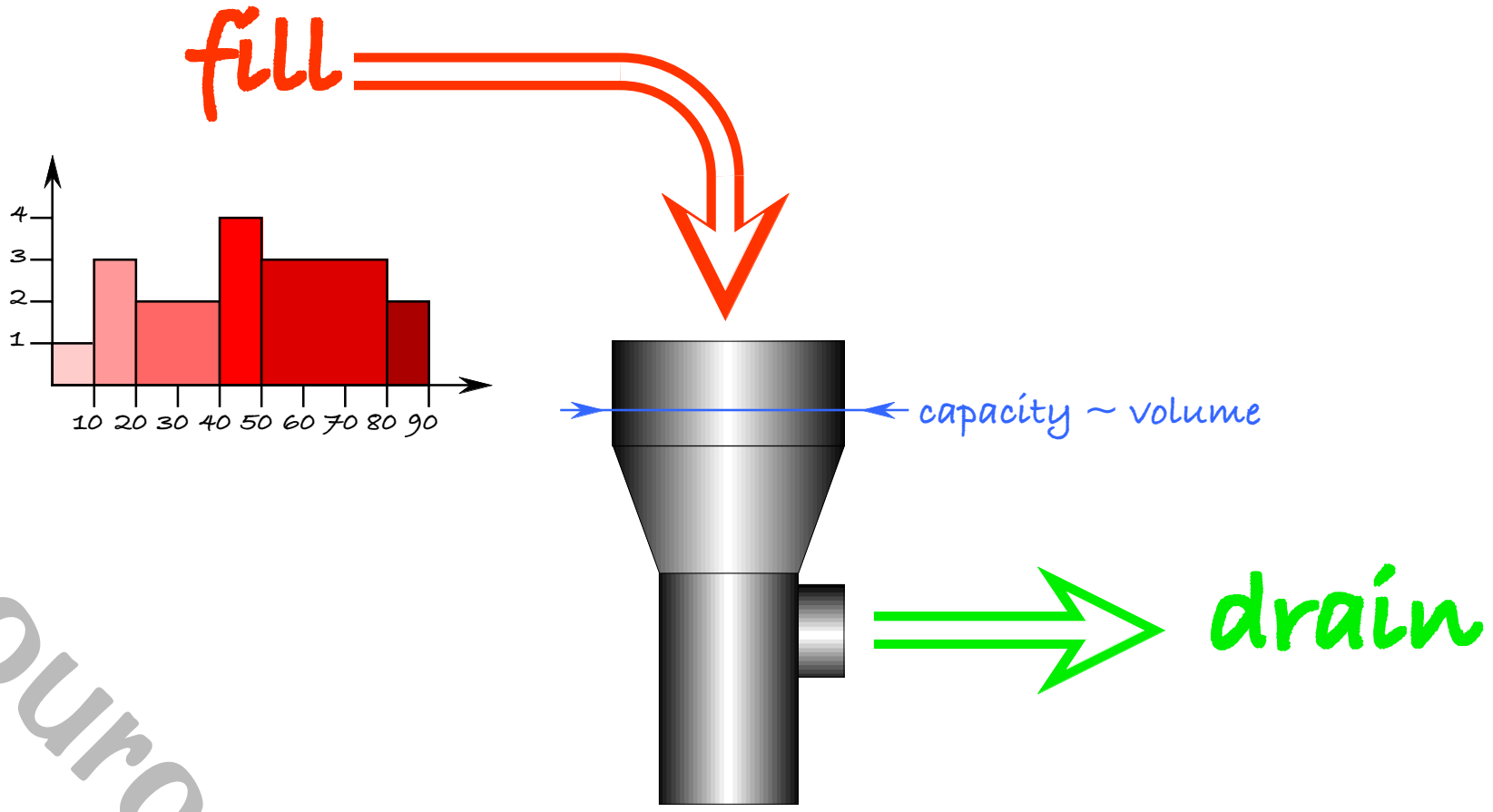
pipe



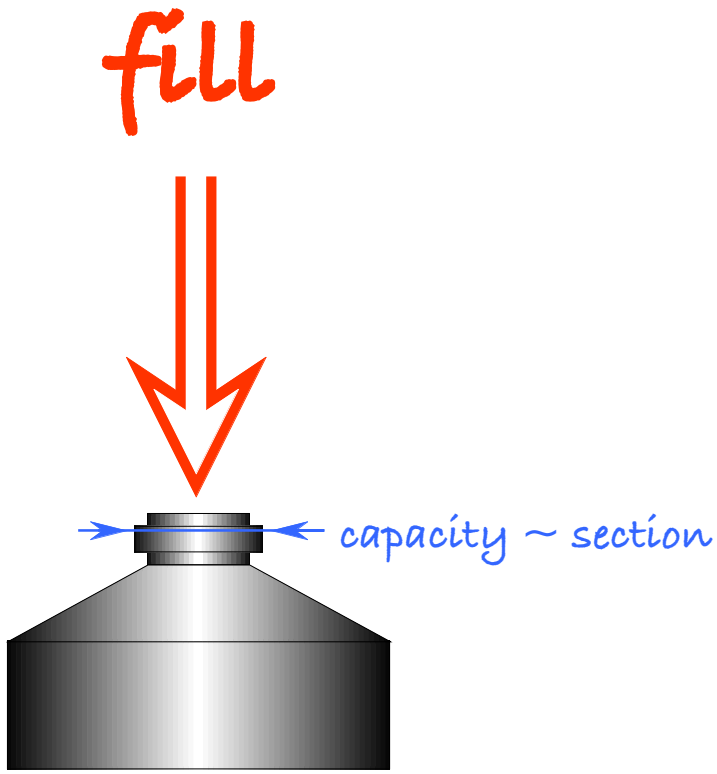
Conduits: components (cont'd)



Conduits: components (cont'd)



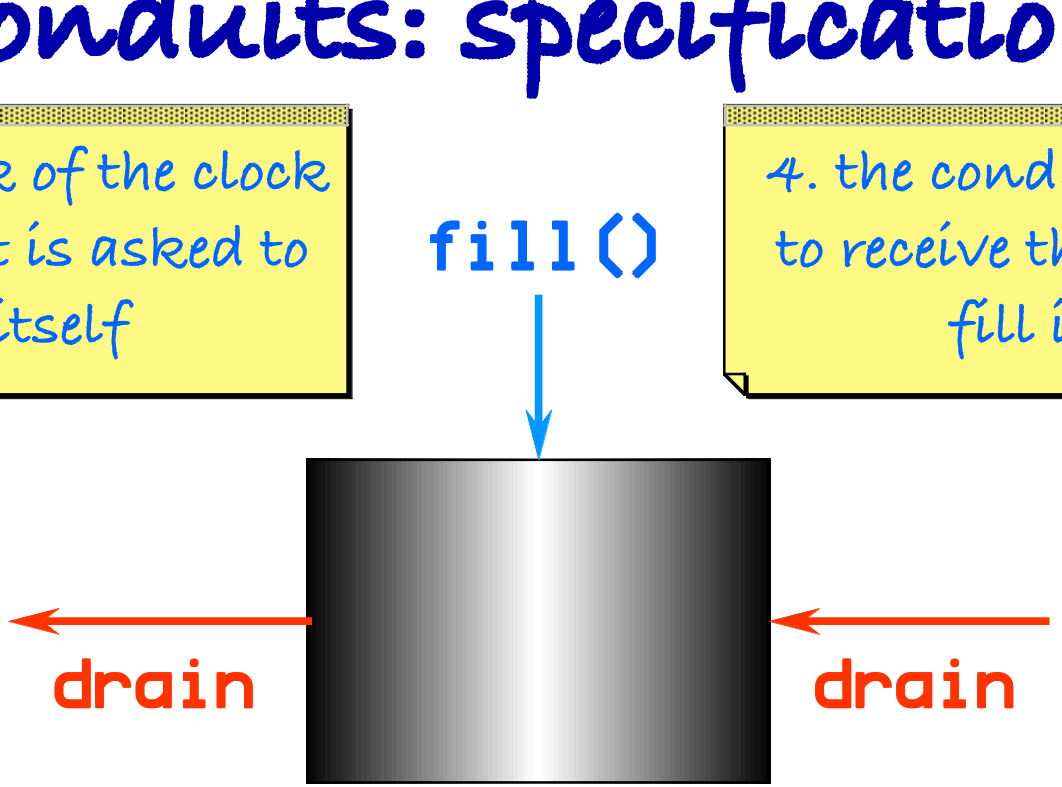
Conduits: components (cont'd)



Conduits: specification

1. every tick of the clock the conduit is asked to fill itself

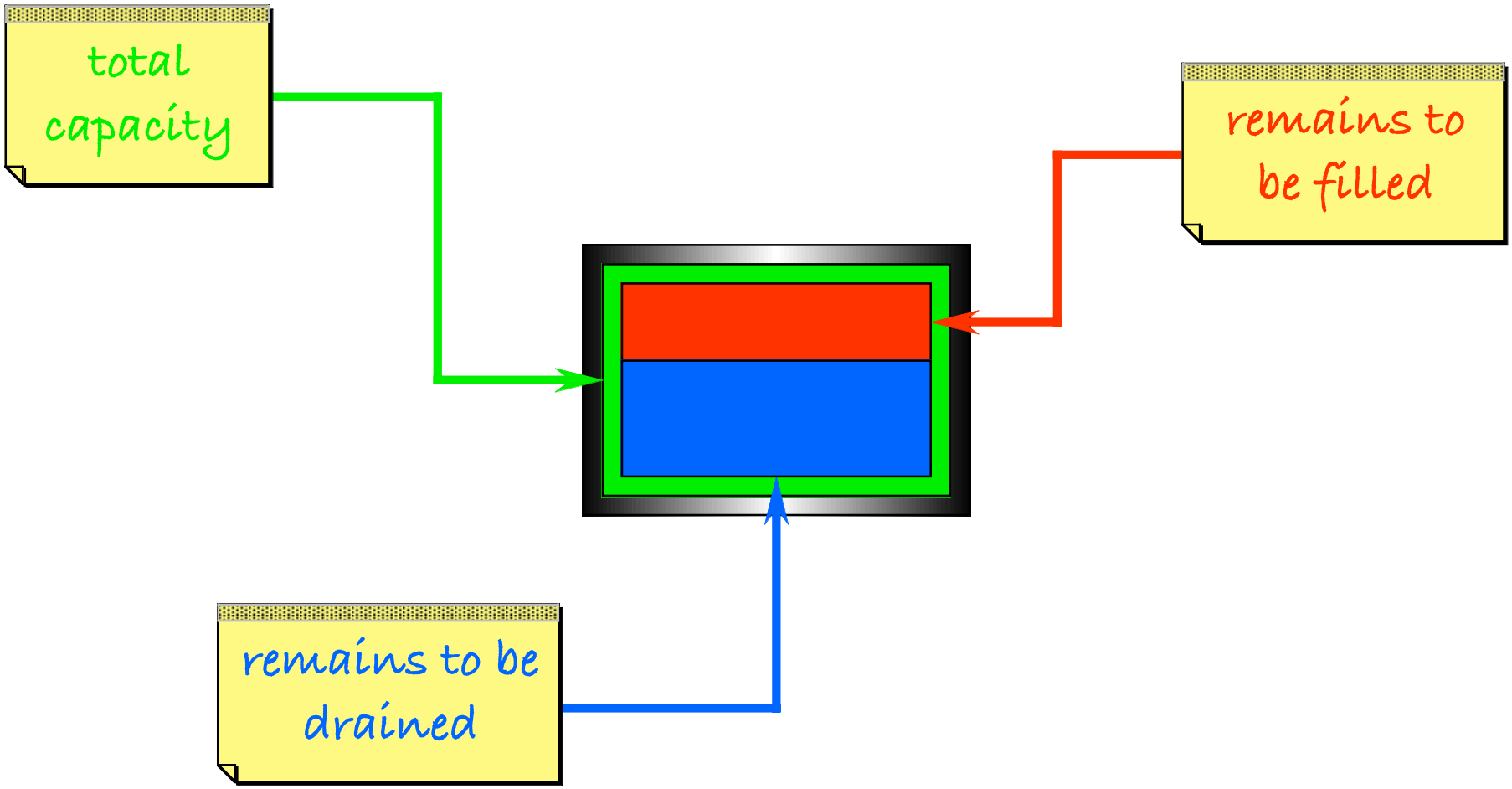
4. the conduit is ready to receive the signal to fill itself



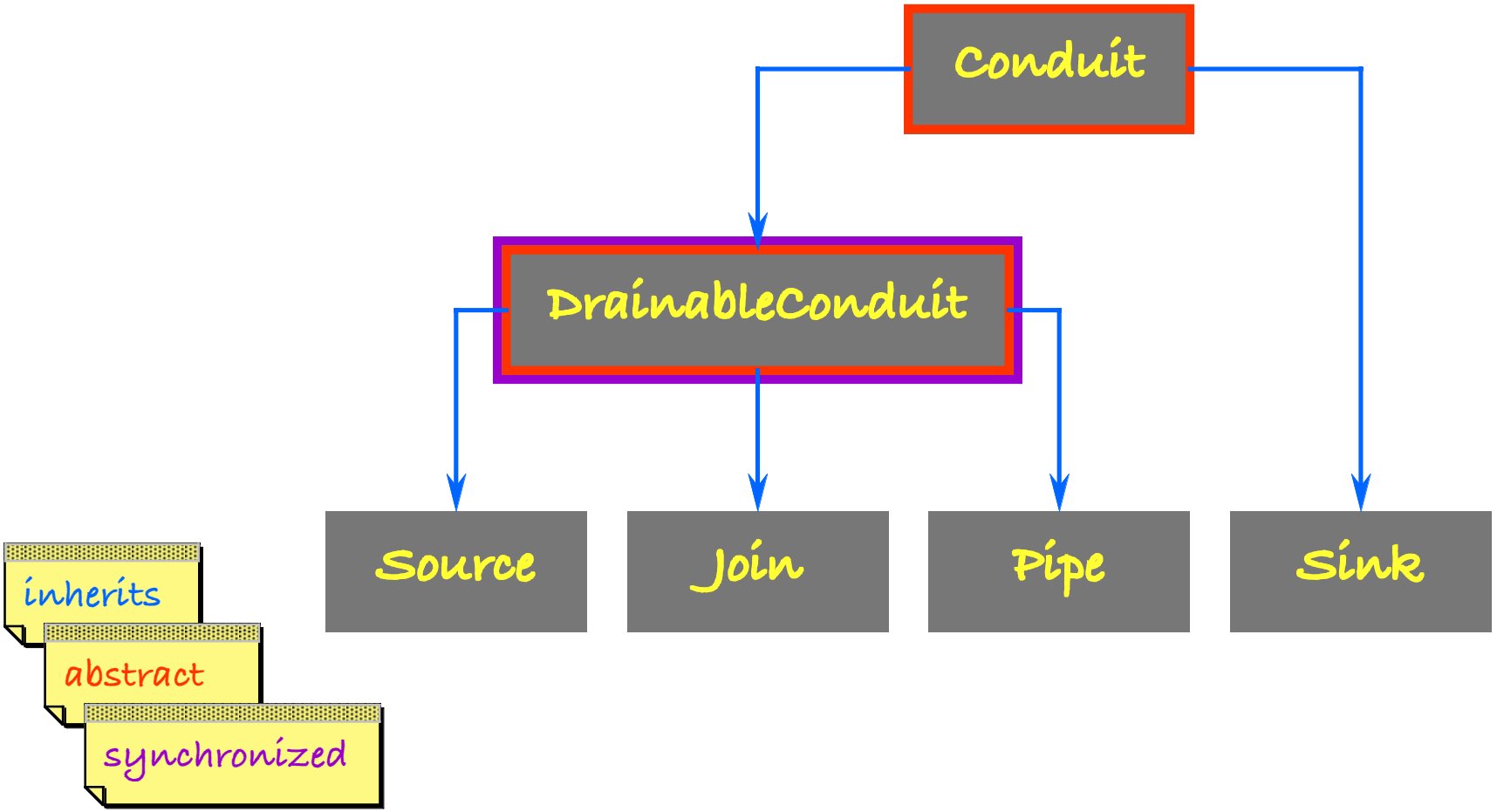
2. non-source conduits are filled by draining other conduits

3. non sink-conduits wait until they are drained

Conduits: specification (cont'd)



Conduits: implementation



Conduits: implementation (cont'd)

```
class Conduit
   variable name
   variable capacity
   variable time is 0
   variable content is 0

   make nam cap is
      name becomes nam
      capacity becomes cap
      while true tick

   method report message value is ...

   method error message is ...

   method fill is
      error "not my job"

   method tick is
      flow is fill;
      content add flow;
      time add 1
      report "filled  " flow
      report "contains" content
```

Conduits: implementation (cont'd)

```
class Conduit
   variable name
   variable capacity
   variable time is 0
   variable content is 0

   make nam cap is
      name becomes nam
      capacity becomes cap
      while true tick

   method report message value is ...

   method error message is ...

   method fill is
      error "not my job"

   method tick is
      flow is fill;
      content add flow;
      time add 1
      report "filled  " flow
      report "contains" content
```

Conduit state

Instantiation

Specialisation

Clock

Conduits: implementation (cont'd)

```
class Sink is Conduit
   variable inConduit

   make nam cap inp is
      inConduit becomes inp
      super make nam cap

   method fill is
      return inConduit drain capacity
```

Conduits: implementation (cont'd)

```
class Sink is Conduit
   variable inConduit

   make nam cap inp is
      inConduit becomes inp
      super make nam cap

   method fill is
      return inConduit drain capacity
```

register input

simple drain

Condu nt'd)

```
class DrainableConduit is Conduit
   variable isFilled is false

   make nam cap is
      super make nam cap

   method tick is
      if isFilled wait
      isFilled becomes true
      super tick
      notify

   method available max is
      if content > max
         return max
      else
         return content

   method drain is
      ifnot isFilled wait
      isFilled becomes false
      flow is available max
      report "drained" flow
      content subtract flow;
      notify
      return flow
```

Condi nt'd)

```
class DrainableConduit is Conduit
   variable isFilled is false

   make nam cap is
      super make nam cap

   method tick is
      if isFilled wait
      isFilled becomes true
      super tick
      notify

   method available max is
      if content > max
       return max
      else
       return content

   method drain is
      ifnot isFilled wait
      isFilled becomes false
      flow is available max
      report "drained" flow
      content subtract flow;
      notify
      return flow
```

overriden clock

default

universal drain

Conduits: implementation (cont'd)

```
class Source is DrainableConduit
   variable profile
   variable produced

   make nam cap pro is
      profile becomes pro
      super make nam cap

   private method lookUp is
      index is 1
      while index ≤ length profile
         if profile[index 1] ≥ time
           return profile[index 2]
      return profile[1 2]

   method fill is
      flow is lookUp
      if flow > capacity
         error "overflow"
      report "produced" flow
      return flow
```

Conduits: implementation (cont'd)

```
class Source is DrainableConduit
   variable profile
   variable produced

   make nam cap pro is
      profile becomes pro
      super make nam cap

   private method lookUp is
      index is 1
      while index ≤ length profile
         if profile[index 1] ≥ time
            return profile[index 2]
      return profile[1 2]

   method fill is
      flow is lookUp
      if flow > capacity
         error "overflow"
      report "produced" flow
      return flow
```

register profile

consult profile

profiled fill

Conduits: implementation (cont'd)

```
class Join is DrainableConduit
   variable inConduit1
   variable inConduit2

   make nam cap in1 in2 is
      inConduit1 becomes in1
      inConduit2 becomes in2
      super make nam cap

   method fill is
      free is (capacity - content) / 2
      flo1 is inConduit1 drain free
      flo2 is inConduit2 drain free
      return flo1 + flo2
```

Conduits: implementation (cont'd)

```
class Join is DrainableConduit
   variable inConduit1
   variable inConduit2

   make nam cap in1 in2 is
      inConduit1 becomes in1
      inConduit2 becomes in2
      super make nam cap

   method fill is
      free is (capacity - content) / 2
      flo1 is inConduit1 drain free
      flo2 is inConduit2 drain free
      return flo1 + flo2
```

register inputs

equitable fill

Condu...t'd)

```
class Pipe is DrainableConduit
   variable inConduit
   variable cell is [0 0 0 0 0 0 0 0 0 0 0]

   make nam cap inc is
      inConduit becomes inc
      super make nam cap

   method available max is
      if cell[10] > max
         flow is max
      else
         flow is cell[10]
      cell[10] subtract flow
      return flow

   method fill is
      free is capacity - cell[1]
      flow is inConduit drain free
      cell[1] add flow
      index is 10
      while index > 1
         free becomes capacity - cell[index]
         available is cell[index - 1]
```

Condu…

```
class Pipe is DrainableConduit
   variable inConduit
   variable cell is [0 0 0 0 0 0 0 0 0 0 0]

   make nam cap inc is
      inConduit becomes inc
      super make nam cap

   method available max is
      if cell[10] > max
         flow is max
      else
         flow is cell[10]
      cell[10] subtract flow
      return flow

   method fill is
      free is capacity - cell[1]
      flow is inConduit drain free
      cell[1] add flow
      index is 10
      while index > 1
         free becomes capacity - cell[index]
         available is cell[index - 1]
```

Cells

register input

Specialisation

Conduits: implementation (cont'd)

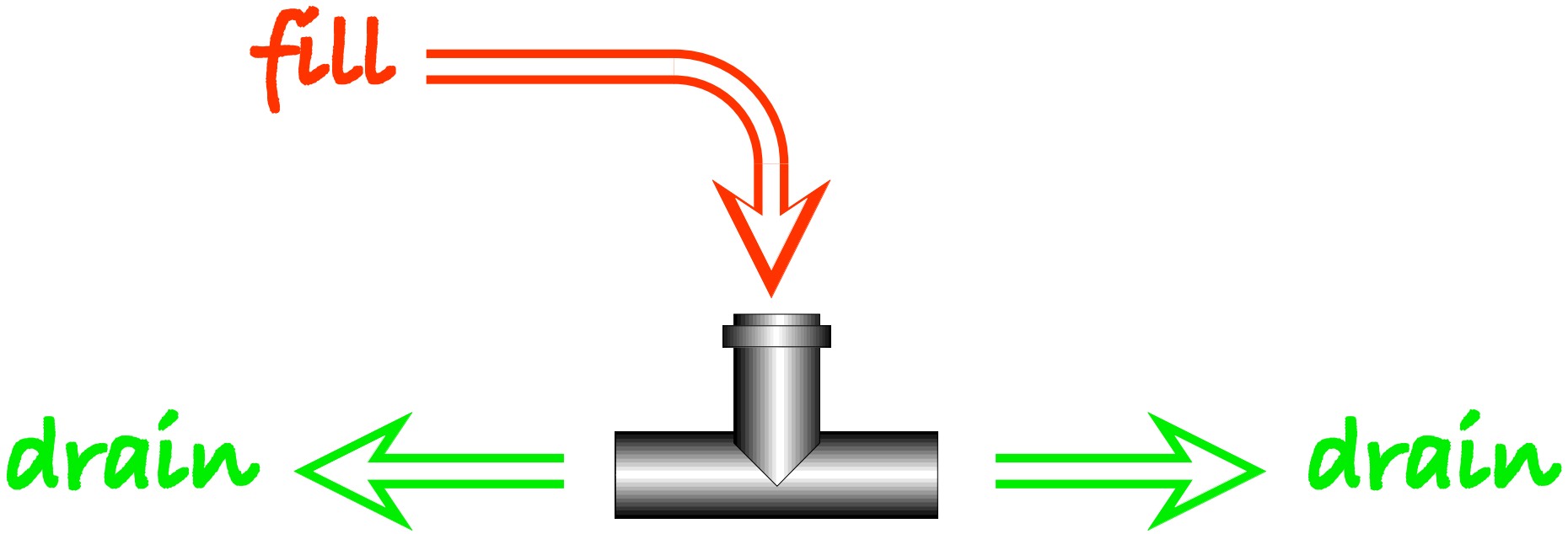
```
      cell[10] subtract flow
   return flow

method fill is
   free is capacity - cell[1]
   flow is inConduit drain free
   cell[1] add flow
   index is 10
   while index > 1
      free becomes capacity - cell[index]
      available is cell[index - 1]
      if available > free
         available becomes free
      cell[index] add available
      index subtract 1
      cell[index] subtract available
   return flow
```

Conduits: implementation (cont'd)

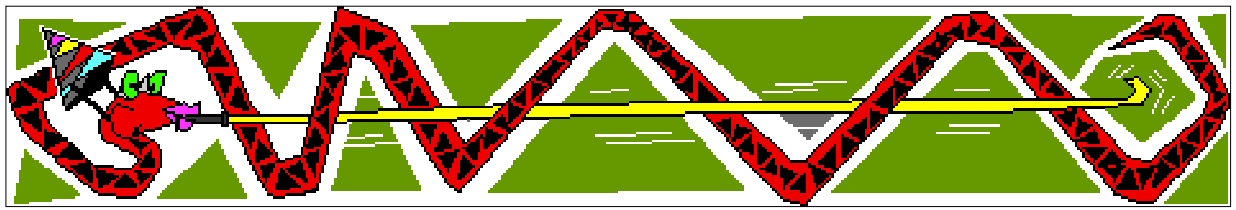
```
      cell[10] subtract flow
   return flow

method fill is
   free is capacity - cell[1]
   flow is inConduit drain free
   cell[1] add flow
   index is 10
   while index > 1
      free becomes capacity - cell[index]
      available is cell[index - 1]
      if available > free
         available becomes free
      cell[index] add available
      index subtract 1
      cell[index] subtract available
   return flow
```

propagated fill

Really hard assignment



hint: view a fork as a composite of two conduits



Appendix

```
package Conduits;

abstract class Conduit extends Thread
  { private String name;
    protected  int capacity;
    protected  int time = 0;
    protected  int content = 0;

    protected Conduit(String nam, int cap)
      { name = nam;
        capacity = cap; }

    final protected void report(String msg, int vol)
      { System.out.println('t=\t' + time + '\t' + name +
                           '\t' + msg + '\t' + vol); }

    final protected void error(String msg)
      { System.out.println(name + '\t=>\t' + msg);
        stop(); }

    abstract protected int fill();

    protected void tick()
      { int flow = fill();
        content += flow;
        time++;
        report("filled    ", flow);
        report("contains  ", content);  }

    final public void run()
      { while (time < 50)
          { tick();
            yield(); }}}
```

```
package Conduits;

abstract class DrainableConduit extends Conduit
  { private boolean isFilled = false;

    protected DrainableConduit(String nam, int cap)
      { super(nam, cap); }

    synchronized protected void tick()
      { if (isFilled)
          try
            { wait(); }
          catch (InterruptedException e)
            { error(e.toString()); }
        isFilled = true;
        super.tick();
        notify(); }

    protected int available(int max)
      { if (content > max)
          return max;
        else
          return content; }

    synchronized final public int drain(int max)
      { int flow;
        if (!isFilled)
          try
            { wait(); }
          catch (InterruptedException e)
            { error(e.toString()); }
        flow = available(max);
        isFilled = false;
        report("drained   ", flow);
        content -= flow;
        notify();
        return flow; }}
```

Appendix (cont'd)

```
package Conduits;

final public class Source extends DrainableConduit
  { private int profile [][];
    private int produced;

    public Source(String nam, int cap, int[][] pro)
      { super(nam, cap);
        profile = pro;
        produced = 0; }

    private int lookUp()
      { for (int index = 0; index < profile.length; index++)
          if (profile[index][0] >= time)
            return profile[index][1];
        return profile[0][1]; }

    protected int fill()
      { int flow = lookUp();
        if (flow > capacity)
          error("overflow");
        report("produced  ", flow);
        return flow; }}
```

```
package Conduits;

import Conduits.*;

final public class Pipe extends DrainableConduit
  { final static int pipeLength = 10;

    private DrainableConduit inConduit;
    private int cell[] = new int[pipeLength];

    public Pipe(String nam, int cap, DrainableConduit inc)
      { super(nam, cap);
        inConduit = inc;
        for(int index = 0; index < pipeLength; index++)
          cell[index] = 0; }

    protected int fill()
      { int flow = inConduit.drain(capacity - cell[0]);
        int free;
        int available;
        cell[0] += flow;
        for(int index = pipeLength - 1; index > 0; index--)
          { free = capacity - cell[index];
            available = cell[index - 1];
            if (available > free)
              available = free;
            cell[index] += available;
            cell[index - 1] -= available; }
        return flow; }

    protected int available(int max)
      { int flow;
        if (cell[pipeLength - 1] > max)
          flow = max;
        else
          flow = cell[pipeLength - 1];
        cell[pipeLength - 1] -= flow;
        return flow; }}
```

Appendix (cont'd)

```
package Conduits;

import Conduits.*;

final public class Join extends DrainableConduit
  { private DrainableConduit inConduit1;
    private DrainableConduit inConduit2;

    public Join(String nam, int cap, DrainableConduit in1,
                                     DrainableConduit in2)
      { super(nam, cap);
        inConduit1 = in1;
        inConduit2 = in2; }

    protected int fill()
      { int free = (capacity - content) / 2;
        int flo1 = inConduit1.drain(free);
        int flo2 = inConduit2.drain(free);
        return (flo1 + flo2); }}
```

```
package Conduits;

import Conduits.*;

final public class Sink extends Conduit
  { private DrainableConduit inConduit;

    public Sink(String nam, int cap, DrainableConduit inc)
      { super(nam, cap);
        inConduit = inc; }

    protected int fill()
      { return inConduit.drain(capacity); }}
```

```
import Conduits.*;
import java.applet.Applet;

public class Simulation extends Applet
  { public static void main(String argv[])
      { final int PRO_1[][] = {{10, 1},{20, 3},{40, 1},
                               {50, 4},{80, 1},{90, 5}};
        final int PRO_2[][] = {{10, 2},{20, 4},{40, 1},
                               {50, 2},{80, 1},{90, 4}};

        Source SRC_1 = new Source ("Source 1", 5, PRO_1);
        Source SRC_2 = new Source ("Source 2", 4, PRO_2);
        Pipe   PIP_1 = new Pipe   ("Pipe   1", 4, SRC_1);
        Pipe   PIP_2 = new Pipe   ("Pipe   2", 2, SRC_2);
        Join   JOI_1 = new Join   ("Join   1", 4, PIP_1,
                                                  PIP_2);
        Pipe   PIP_3 = new Pipe   ("Pipe   3", 3, JOI_1);
        Sink   SNK_1 = new Sink   ("Sink   1", 4, PIP_3);

        SRC_1.start();
        SRC_2.start();
        PIP_1.start();
        PIP_2.start();
        JOI_1.start();
        PIP_3.start();
        SNK_1.start(); }}
```