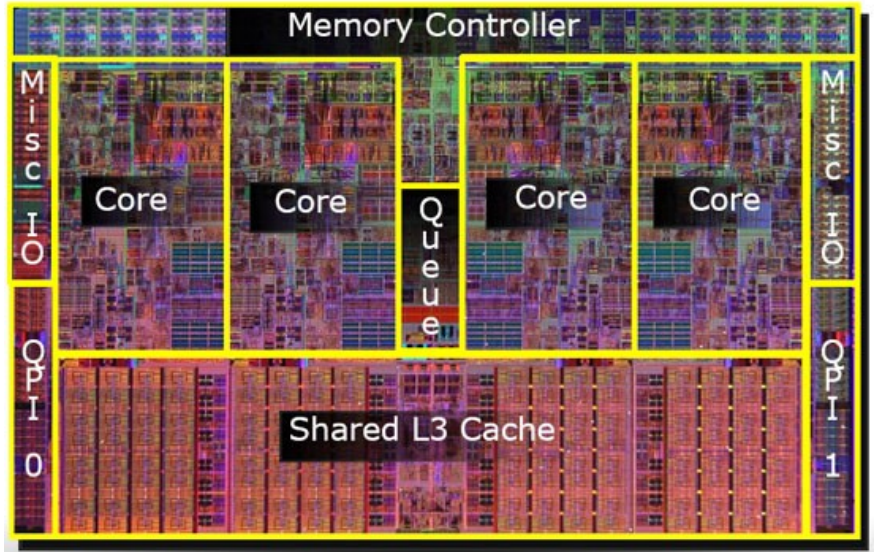
Concurrency

Václav Pech

http://jroller.com/vaclav

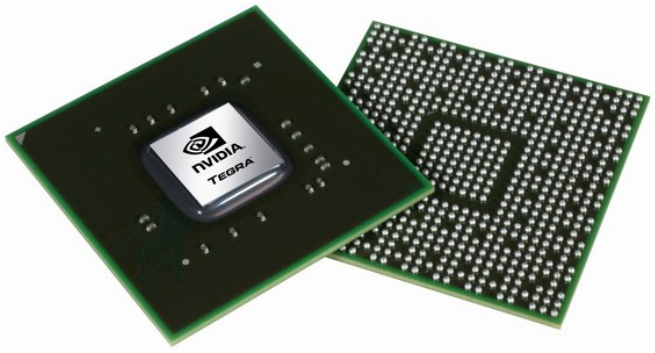
http://www.vaclavpech.eu

@vaclav_pech

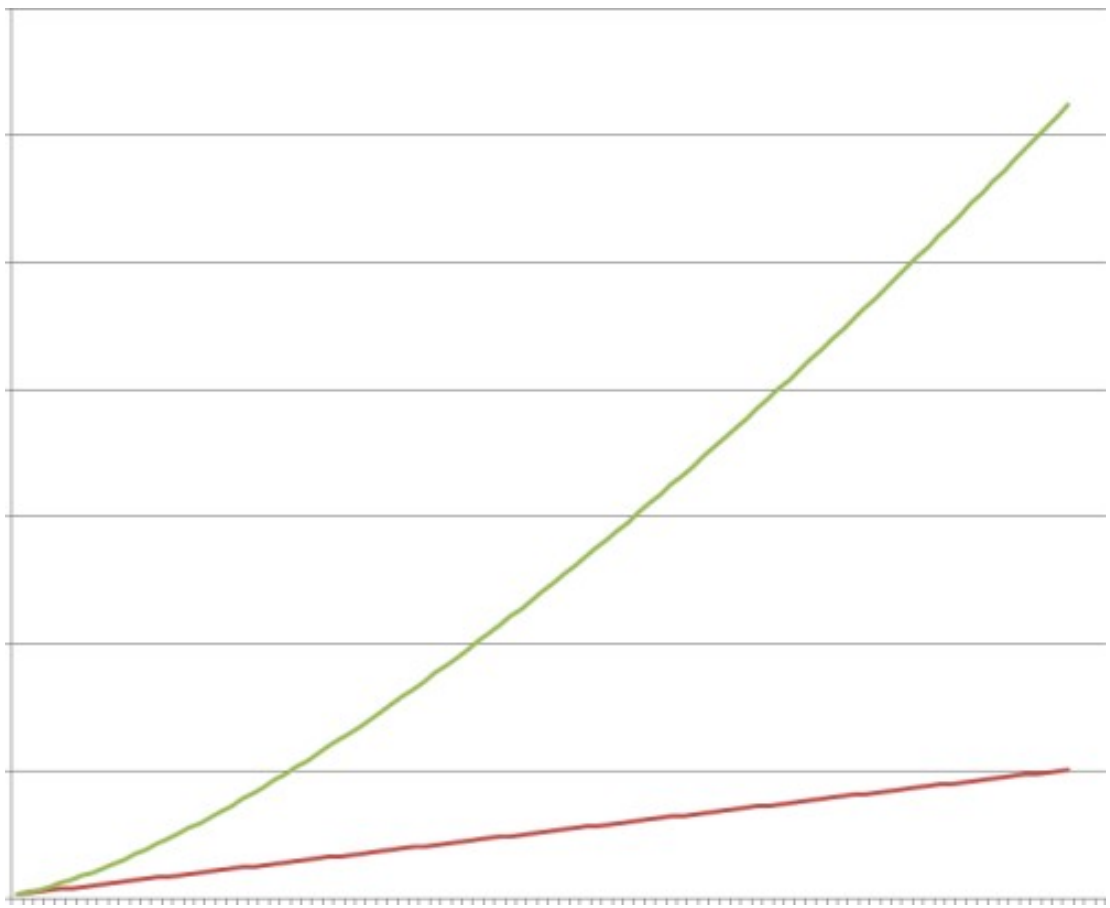


We're all in the parallel computing business!

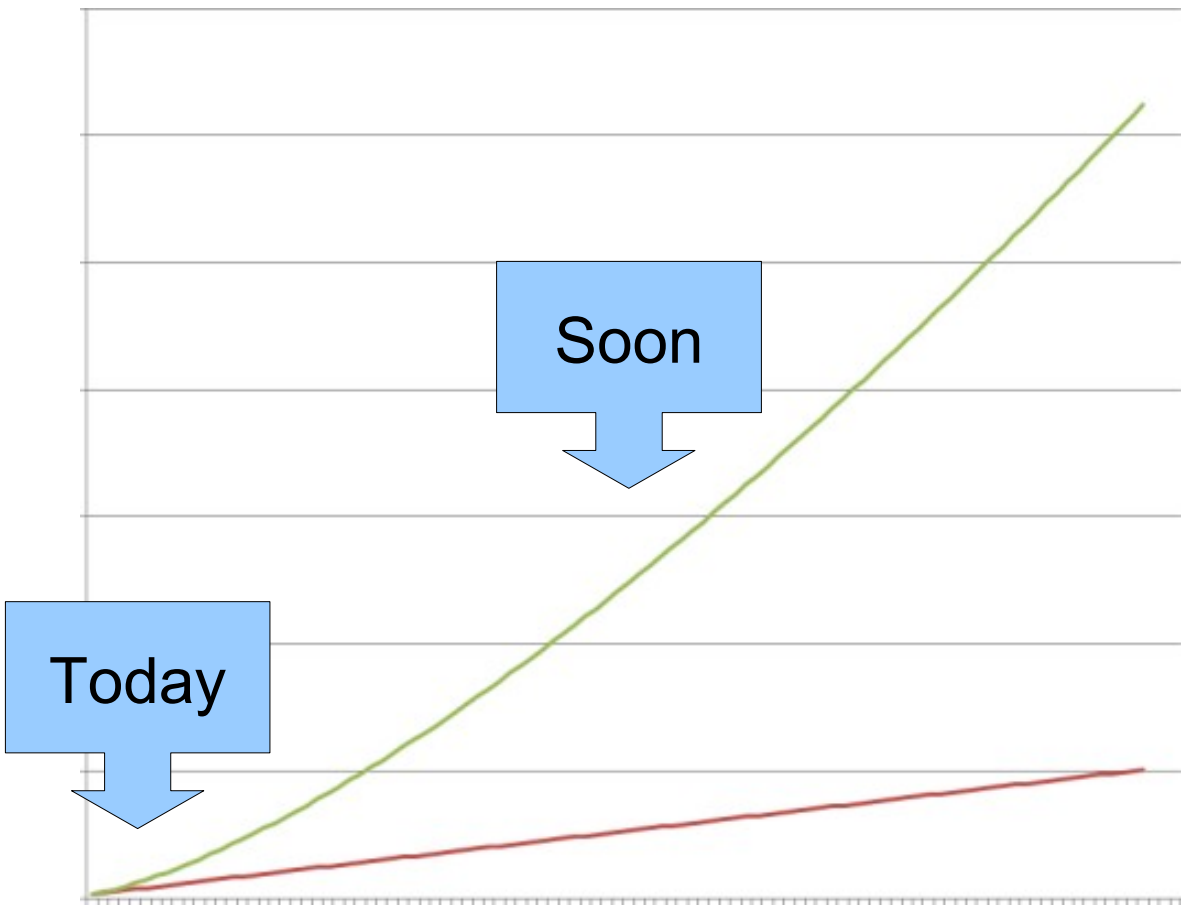

LG
5:56 PM
London
12/13 2010 MON
05 56
6
Browser
Gmail
Market
Camera
Phone
Contacts
Messaging
Applicati...


NVIDIA
TEGRA

of cores



of cores



Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {

        count++;

    }
}
```

Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this) {
            count++;
        }
    }
}
```

Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this.getClass()) {
            count++;
        }
    }
}
```

Dealing with threads sucks!

```
public class ClickCounter implements ActionListener {
    public ClickCounter(JButton button) {
        button.addActionListener(this);
    }

    public void actionPerformed(final ActionEvent e) {
        ...
    }
}
```

Stone age of parallel SW

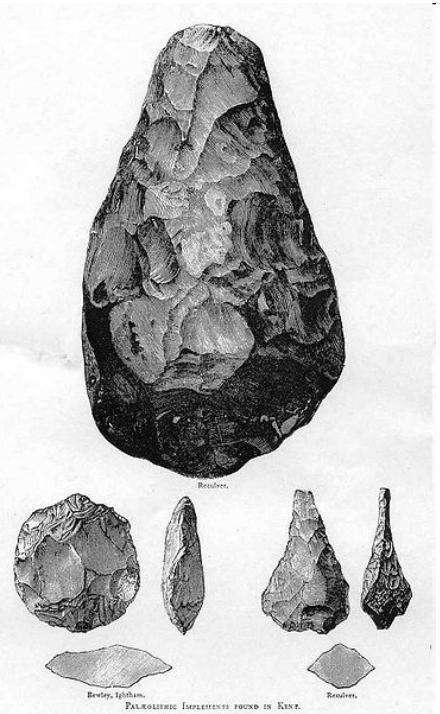
Dead-locks

Live-locks

Race conditions

Starvation

Shared Mutable State



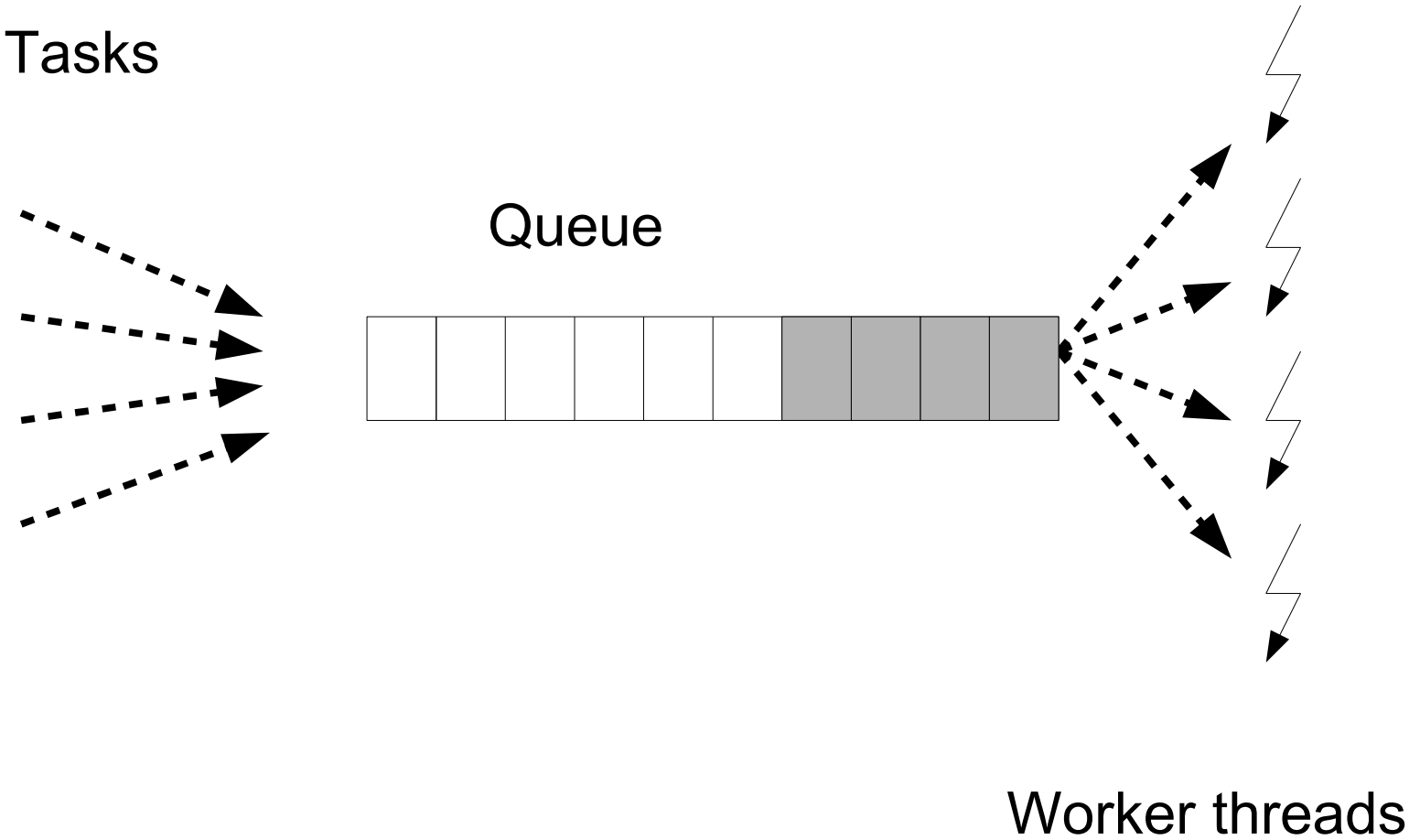
Multithreaded programs today work mostly by accident!

Can we do better?

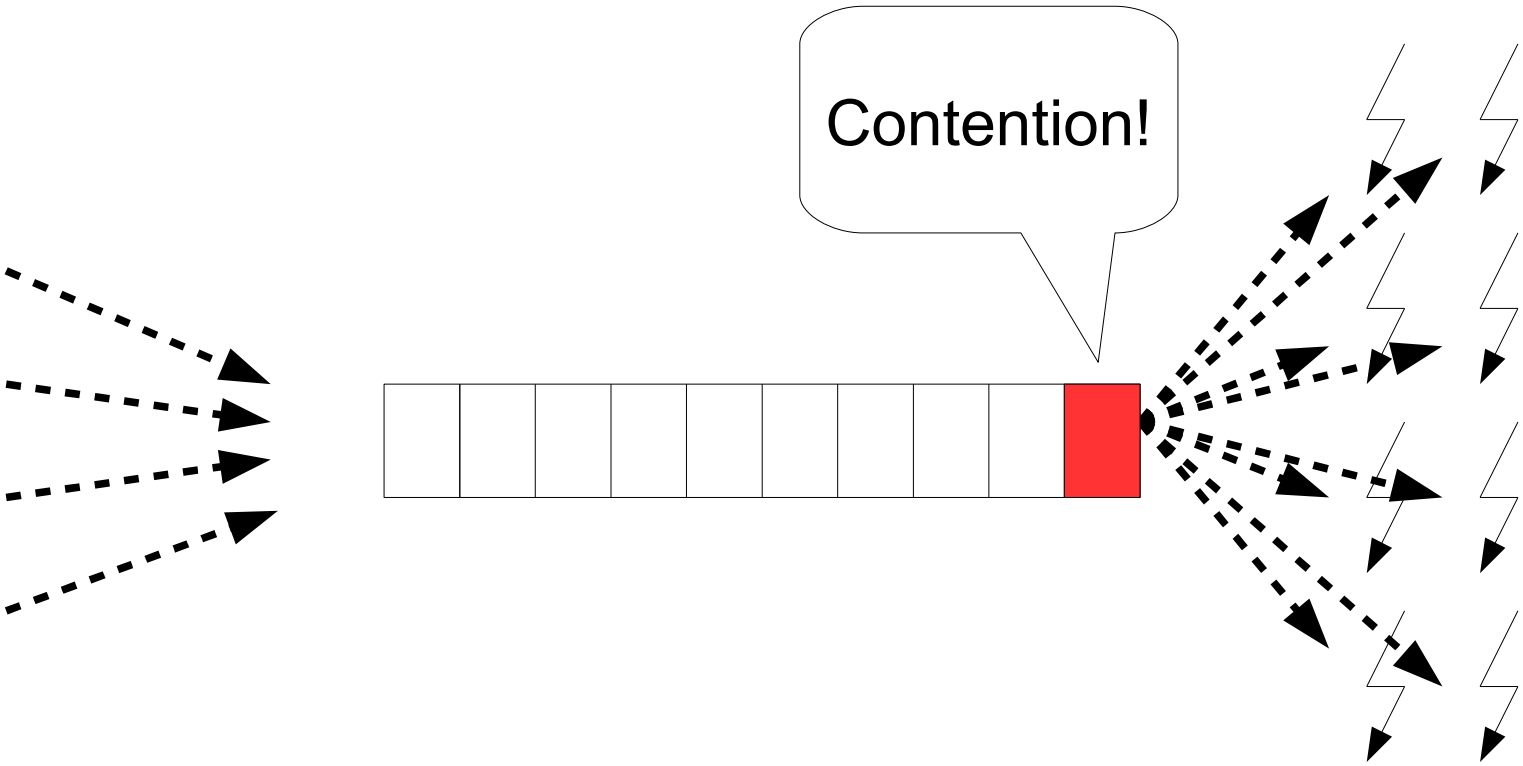
Asynchronous invocation

```
Future f = threadPool.submit(calculation);
…
System.out.println(“Result: “ + f.get());
```

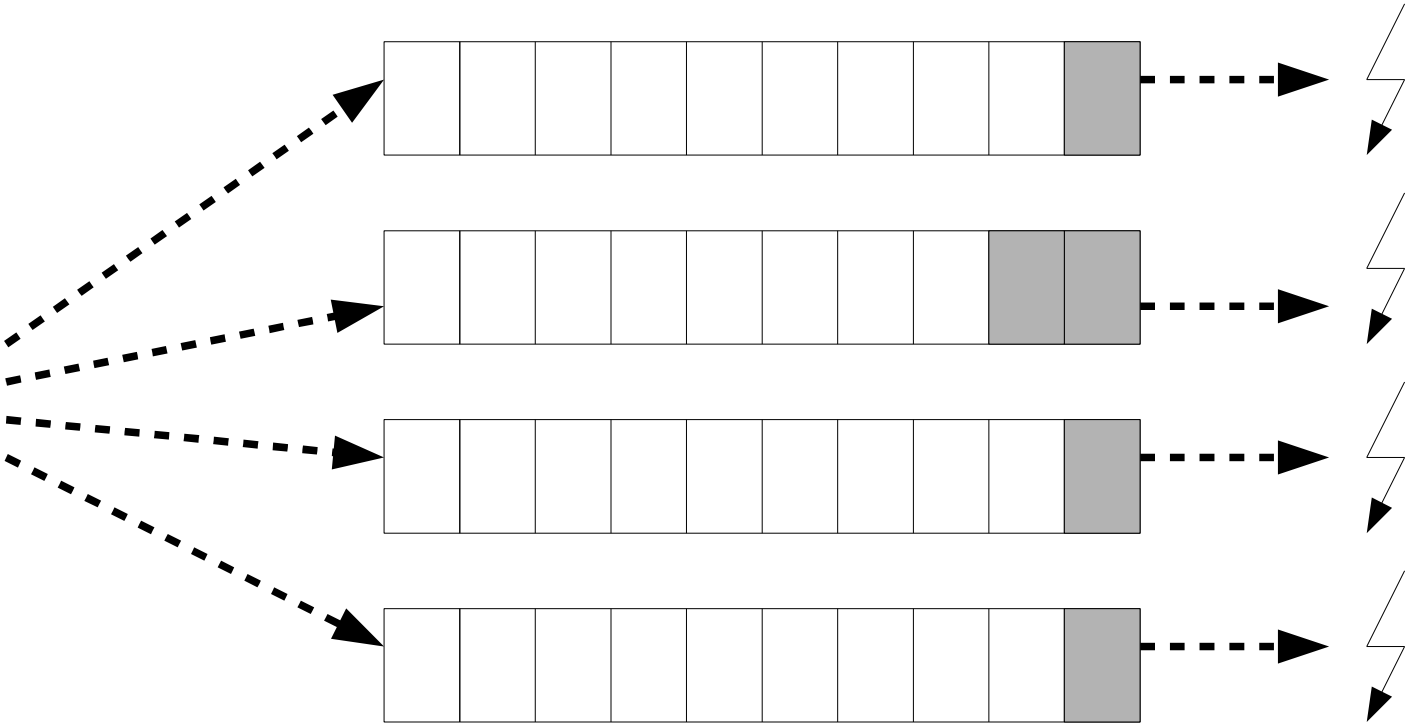
Thread Pool



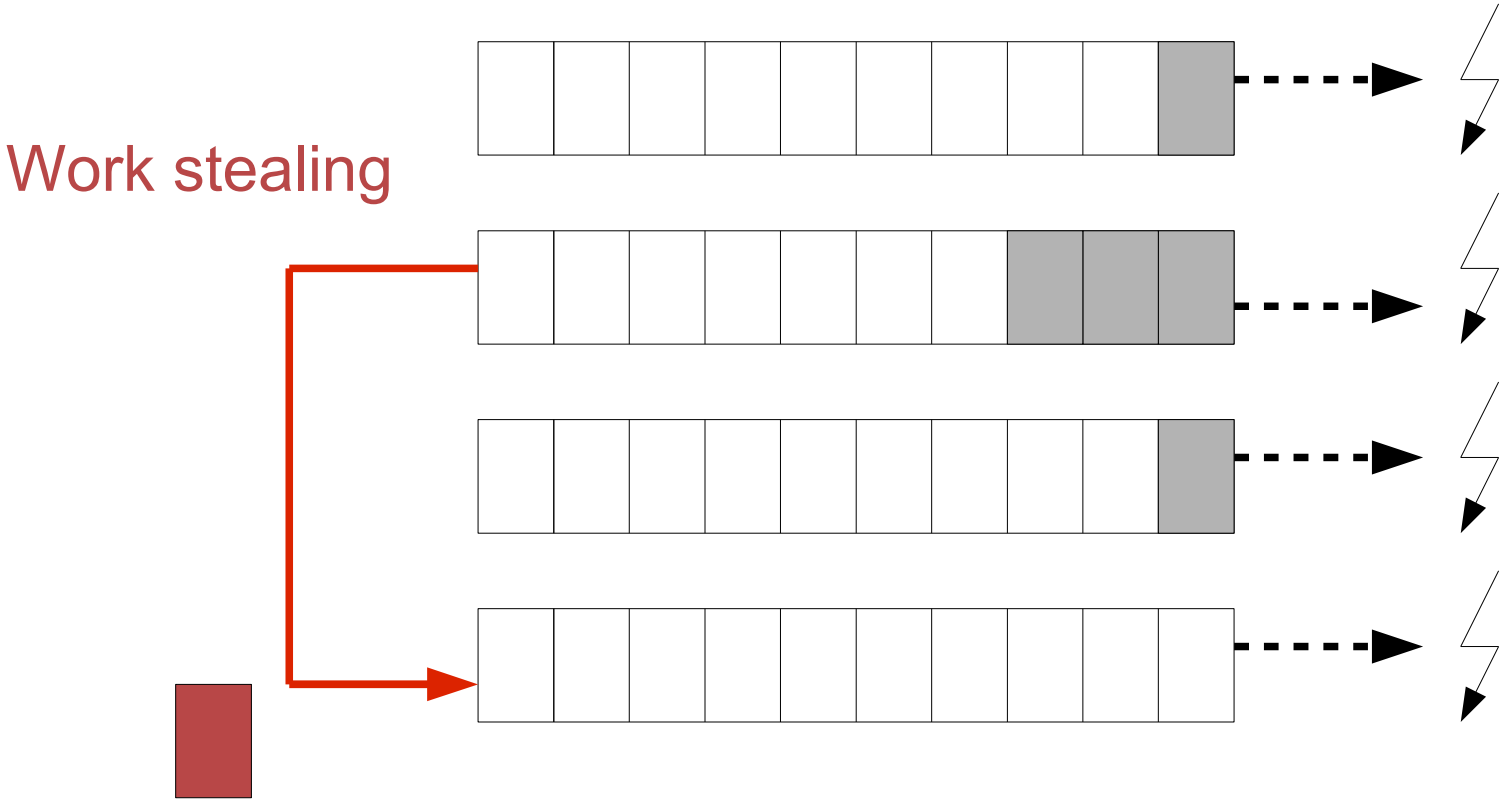
Thread Pool



Fork/Join Thread Pool

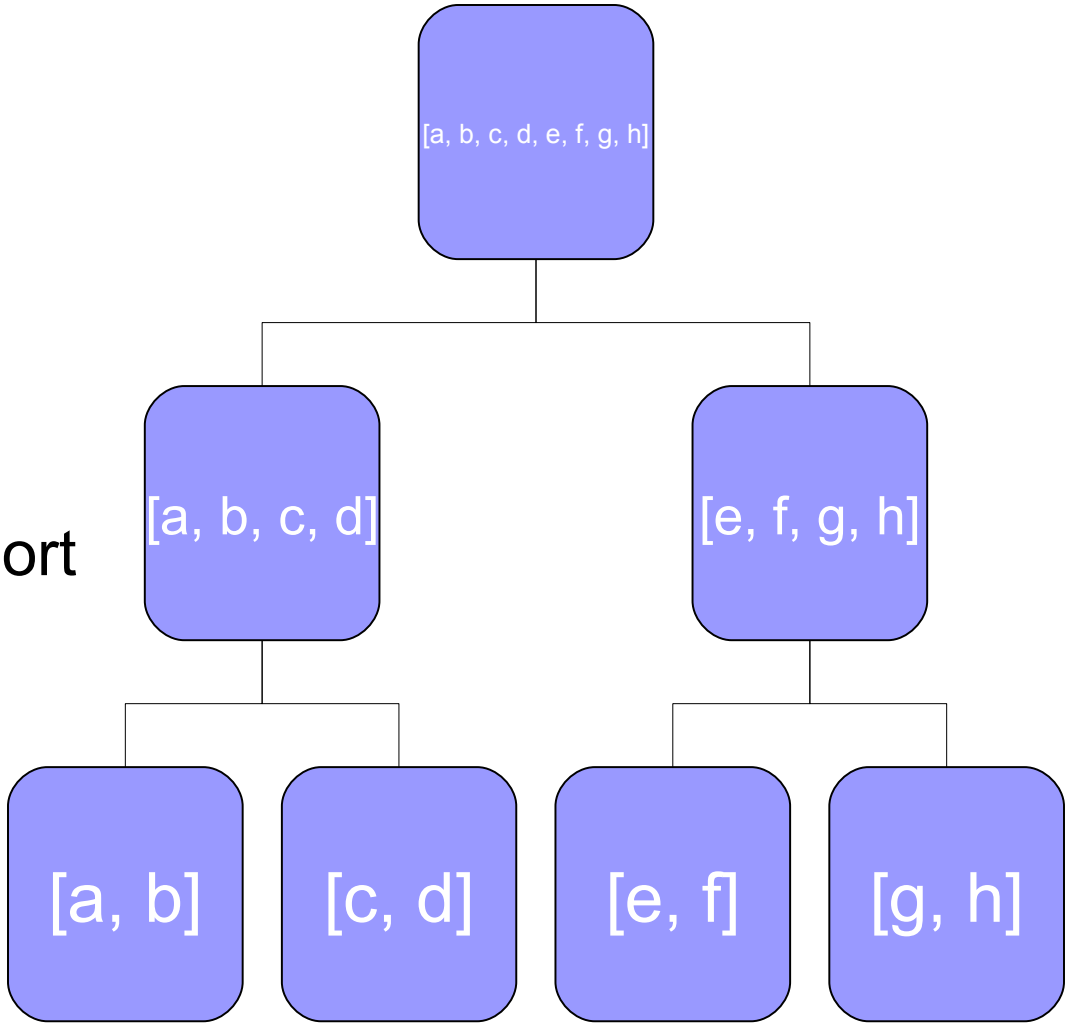


Fork/Join Thread Pool



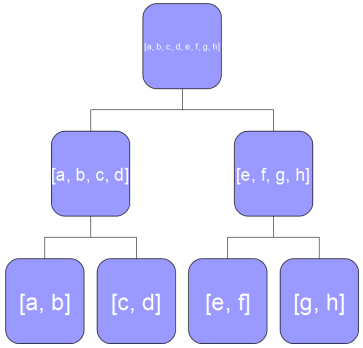
Fork/Join

- Solve hierarchical problems
  - Divide and conquer
  - Merge sort, Quick sort
  - Tree traversal
  - File scan / search
  - …



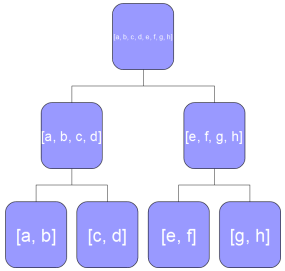
Fork/Join (GPars)

```
runForkJoin(new File("./src")) {currentDir ->
    long count = 0;
    currentDir.eachFile {
        if (it.isDirectory()) {
            forkOffChild it
        } else {
            count++
        }
    }
    return count + childrenResults.sum(0)
}
```



Waits for children without blocking the thread!

Collections (Groovy/GPars)




```
images.eachParallel {it.process()}

documents.sumParallel()

candidates.maxParallel {it.salary}.marry()
```

Parallel Collections

```
progLanguages.parallel.filter {it.concurrent}
                      .max {it.javaInteroperability}
                      .map {it.logo}  ==
```

Parallel Arrays (jsr-166y)

```
ParallelArray namesOfWomen =
    people.withFilter(aWoman).withMapping(retrieveName).all();
```

```
Ops.Predicate aWoman = new Ops.Predicate() {
  public boolean op(Person friend) {return !friend.isMale();}
};

Ops.Op retrieveName = new Ops.Op() {
  public Object op(Person friend) {return friend.getName();}
};
```

Languages are either concurrent or obsolete.

Java 5

Asynchronous calculations

Java 7

Asynchronous calculations

Fork/Join

Java 8

Asynchronous calculations

Fork/Join

Parallel collections

Scala

Asynchronous calculations

Fork/Join

Parallel collections

Actors

Clojure

Asynchronous calculations

Fork/Join

Parallel collections

Actors

Agents, Stm

Oz

Asynchronous calculations

Fork/Join

Parallel collections

Actors

Agents, Stm

Dataflow

Google's Go

Asynchronous calculations

Fork/Join

Parallel collections

Actors

Agents, Stm

Dataflow

CSP

- ✓ Asynchronous calculations
- ✓ Fork/Join
- ✓ Parallel collections
- ✓ Actors
- ✓ Agents, Stm
- ✓ Dataflow
- ✓ CSP

Composing async functions

```
int hash1 = hash(download('http://www.gpars.org'))
int hash2 = hash(loadFile('/gpars/website/index.html'))
boolean result = compare(hash1, hash2)
println result
```

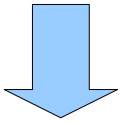
Composing async functions

```
@AsyncFun hash = oldHash
@AsyncFun compare = oldCompare
@AsyncFun download = oldDownload
@AsyncFun loadFile = oldLoadFile

def hash1 = hash(download('http://www.gpars.org'))
def hash2 = hash(loadFile('/gpars/website/index.html'))
def result = compare(hash1, hash2)
println result.get()
```

Composing async functions

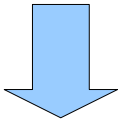
```
@AsyncFun hash = oldHash
@AsyncFun(blocking = true) compare = oldCompare
@AsyncFun download = oldDownload
@AsyncFun loadFile = oldLoadFile

def hash1 = hash(download('http://www.gpars.org'))
def hash2 = hash(loadFile('/gpars/website/index.html'))
boolean result = compare(hash1, hash2)
println result
```

int hash(String text) {…}



Promise<int> hash(*Promise<String>* | *String* text)

int hash(String text) {…}



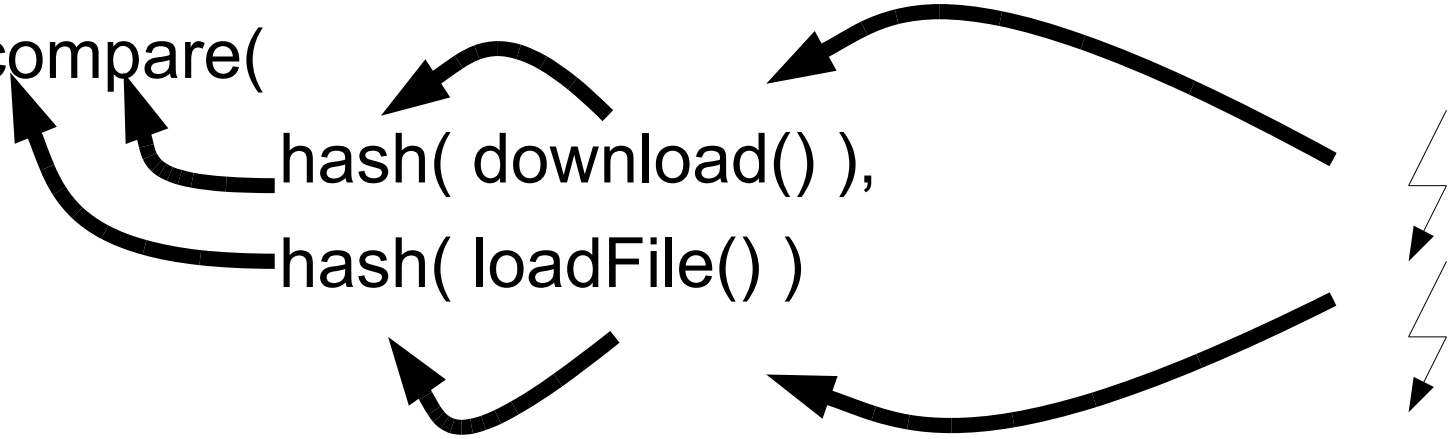
Promise<int> hash(*Promise<String>* | *String* text)

compare(

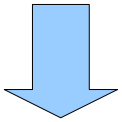
hash(download()),

hash(loadFile())

)



int hash(String text) {…}



Promise<int> hash(*Promise<String>* | *String* text) {

1. Return a Promise for the result
2. Wait (non-blocking) for the text param
3. Call the original *hash()*
4. Bind the result

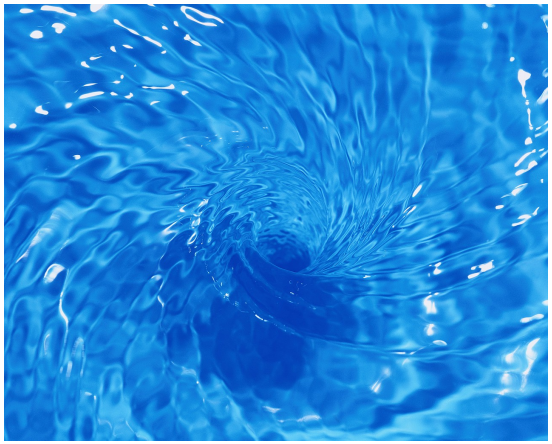
}

Composing async functions

Combine functions as usual

Parallelism is detected automatically

Dataflow Concurrency



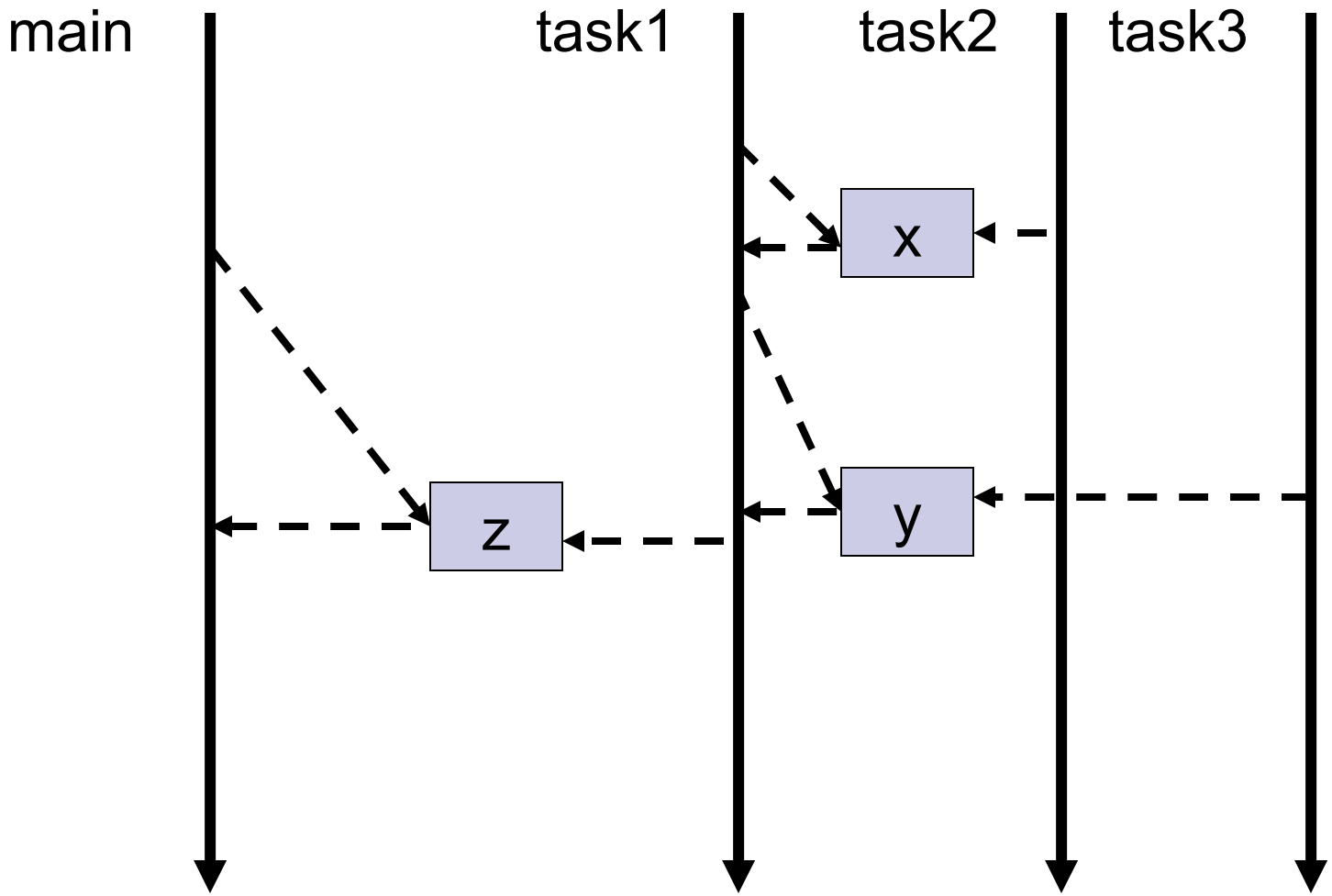
- No race-conditions
- No live-locks
- Deterministic deadlocks

  Completely deterministic programs

BEAUTIFUL code

(Jonas Bonér)

Dataflow Variables / Promises



Dataflows

```
def df = new Dataflows()

task { df.z = df.x + df.y }
task { df.x = 10 }
task {
    println ”I am task 3”
    df.y = 5
}
assert 15 == df.z
```

Milestone

Asynchronous calculations

Fork/Join

Parallel collection processing

Dataflow variables/streams

Message Passing

Actors

Processes with mailboxes

Communicating Sequential Processes (CSP)

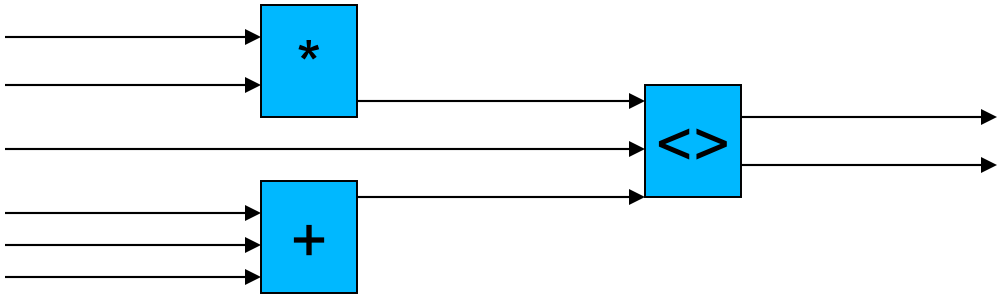
Sequential processes synchronously talking through channels

Dataflow Operators

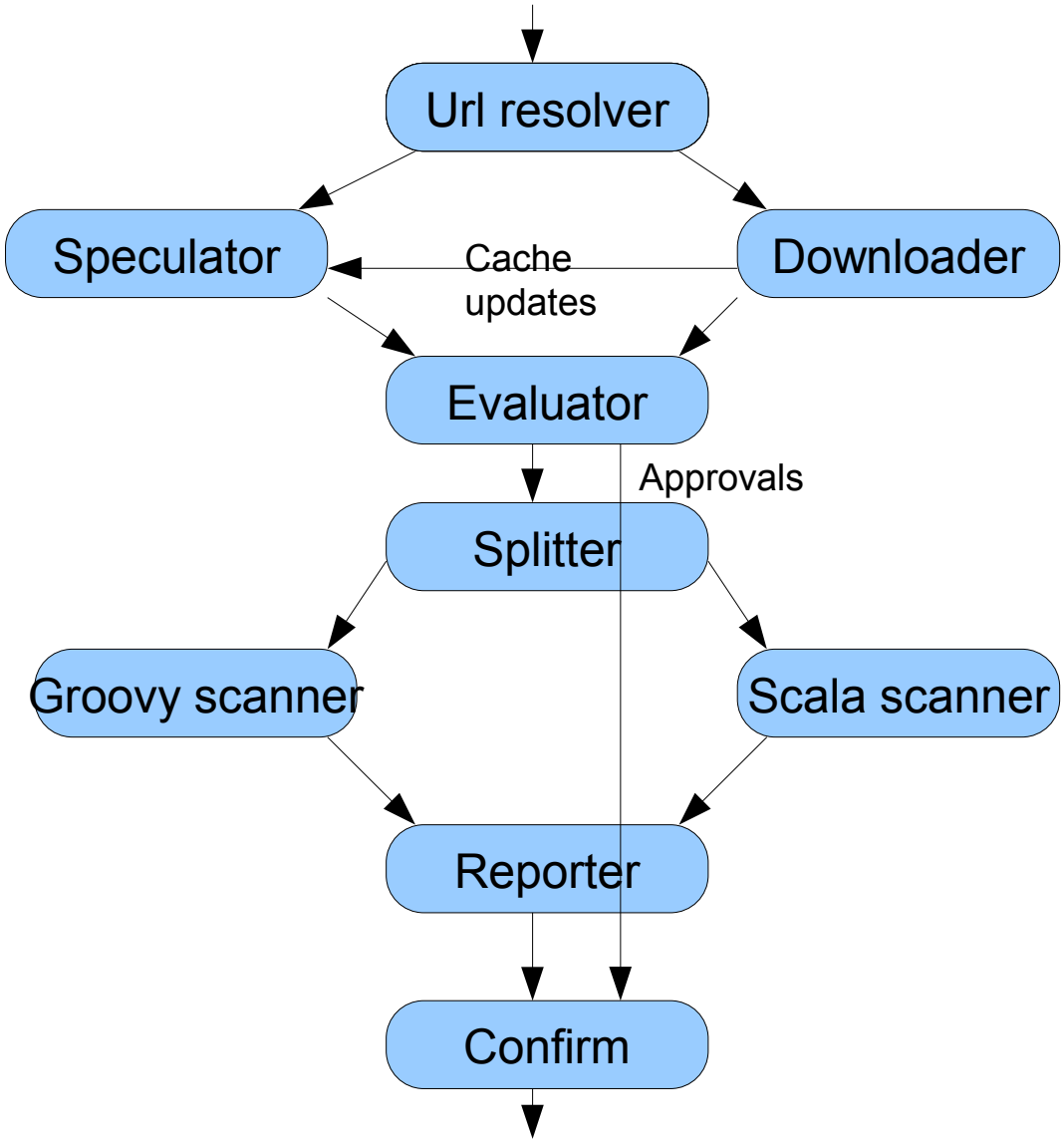
Event-triggered computations connected by channels into a graph

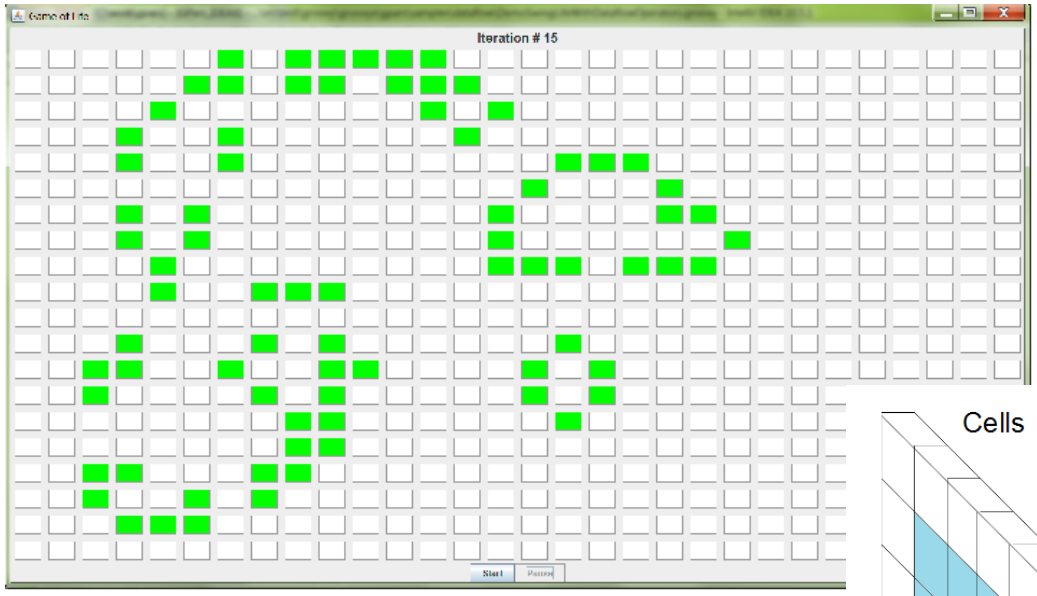
Dataflow Operators

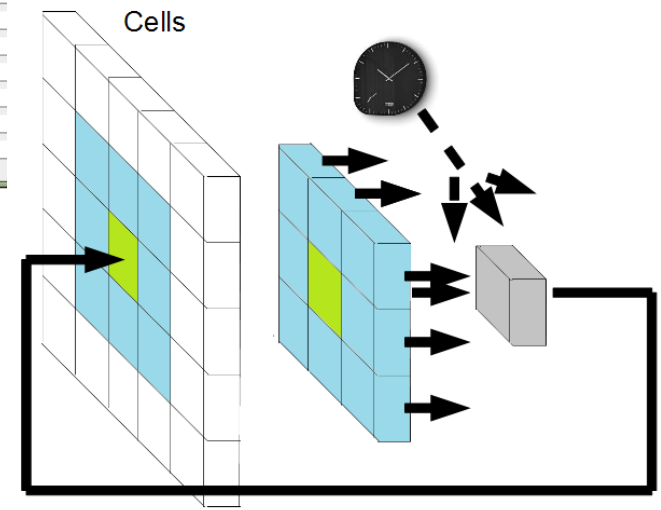
```
operator(inputs: [headers, bodies, footers],
         outputs: [articles, summaries])
  {header, body, footer ->
    def article = buildArticle(header, body, footer)
    bindOutput(0, article)
    bindOutput(1, buildSummary(article))
  }
```



Dataflow Operators

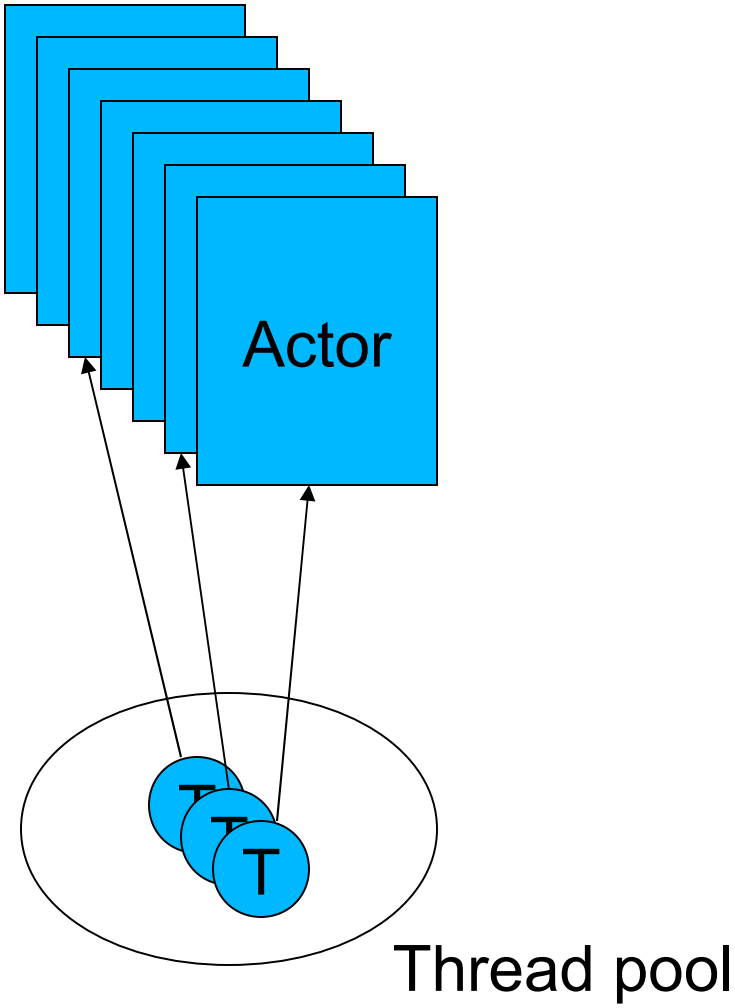



Iteration # 15
Start

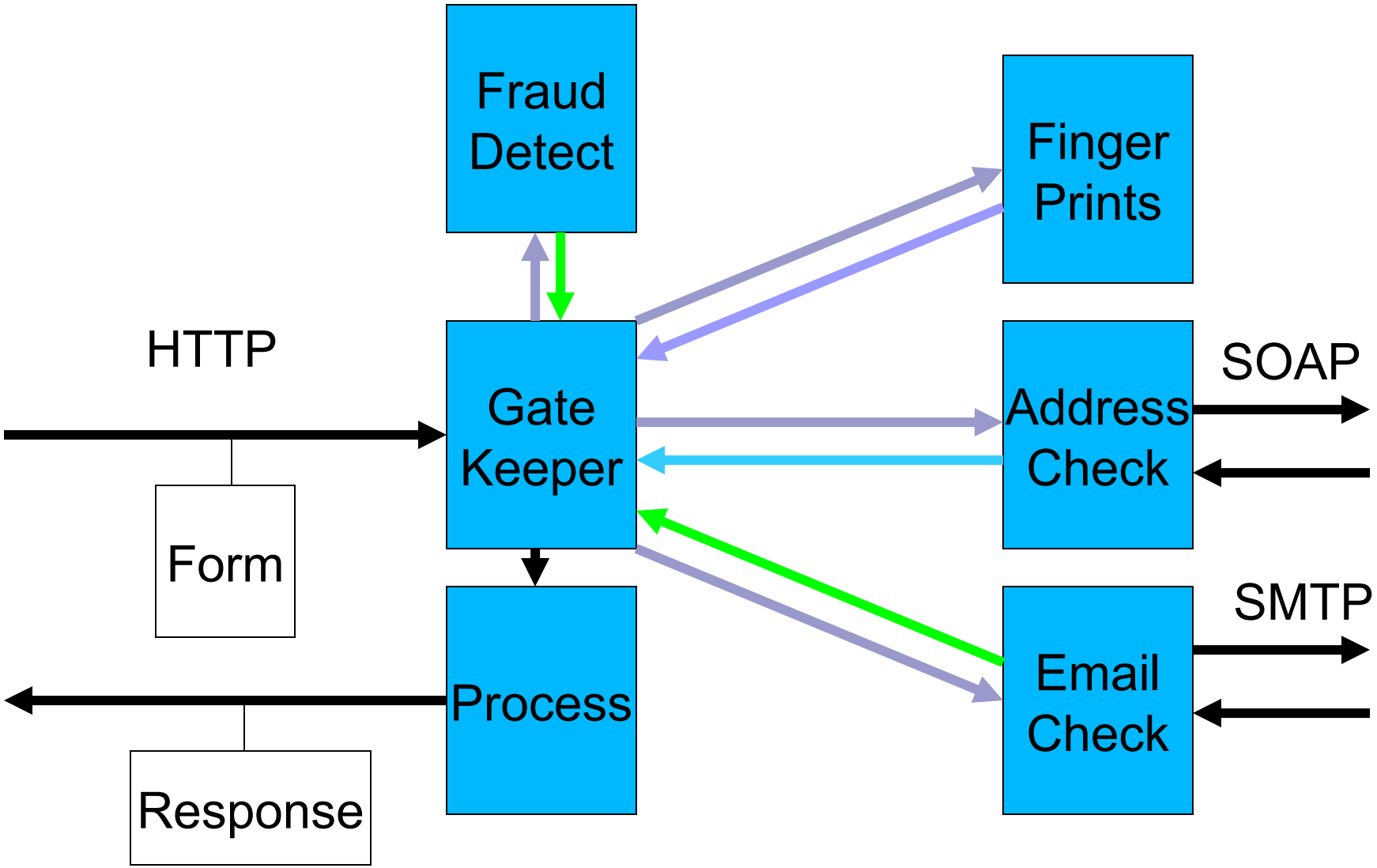

Cells

Actors

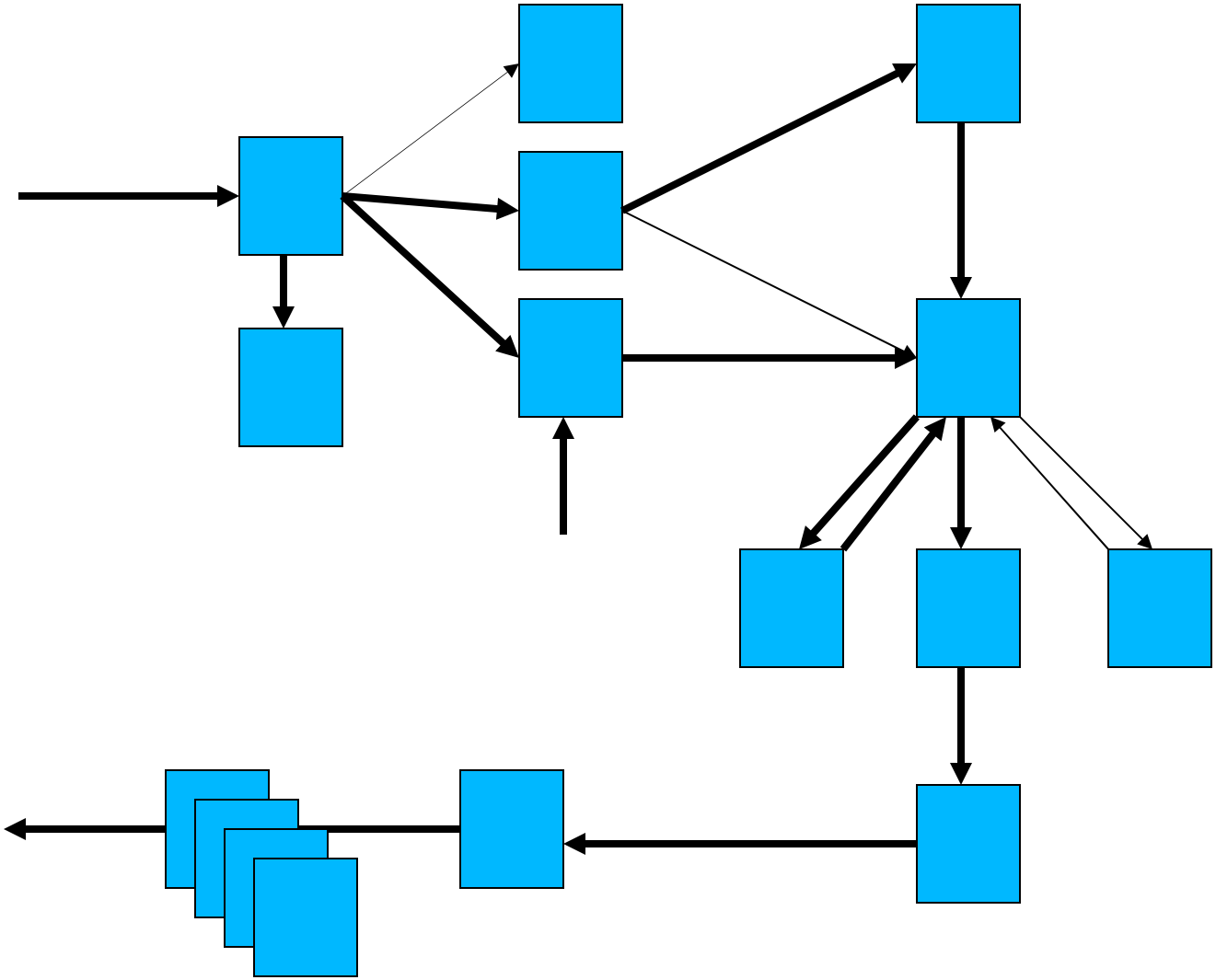
- Isolated
- Communicating
  - Immutable messages
- Active
  - Pooled shared threads
- Activities
  - Create a new actor
  - Send a message
  - Receive a message



Actors use


Fraud Detect
Finger Prints
HTTP
Gate Keeper
Address Check
SOAP
Form
SMTP
Process
Email Check
Response

Actors patterns



Enricher

Router

Translator

Endpoint

Splitter

Aggregator

Filter

Resequencer

Checker

Sending messages

```
buddy.send 10.eur

buddy << new Book(title:'Groovy Recipes',
                  author:'Scott Davis')

def canChat = buddy.sendAndWait 'Got time?'

buddy.sendAndContinue 'Need money!', {cash->
    pocket.add cash
}
```

Stateless Actors (pure Java)

```
class MyActor extends DynamicDispatchActor {
    private Account account = ...
    public void onMessage(String msg) {
        String encripted = encrypt(msg);
        reply(encripted);
    }
    public void onMessage(Integer number) {
        reply(2 * number);
    }

    public void onMessage(Money cash) {
        System.out.println("Received a donation " + cash);
        account.deposit(cash);
    }
}
```

Stateful Actors

```
class MyActor extends DefaultActor {
    void act() {
        def buddy = new YourActor()
        buddy << ‘Hi man, how\’re things?’
        def response = receive()
    }
}
```

Implicit State in Actors

```
val me = actor {

    react {msg1 ->
      switch (msg1) {
        case Work: reply "I don't work so early" ; stop();
        case Breakfast:
          msg1.eat()
            react {msg2 →
              switch (msg2) {
                case Work: reply "OK, time to work"; msg2.do()
                case Lunch: ...
}  }  }     }
```

Continuation Style

```
loop {
    …
    react {
        …
        react {/* schedule the block and exit */
            …
        }
        //Never reached
    }
    //Never reached
}
//Never reached
```

Java actor frameworks

Jetlang
Kilim
ActorFoundry
Actorom
Akka
GPars (Yes!)
…

Actors

Processes with a mail-box

Share no data

Communicate by sending messages

Use a thread-pool

Active Objects

```
@ActiveObject
class MyCounter {
    private int counter = 0

 @ActiveMethod
    def incrementBy(int value) {
        println "Received an integer: $value"
        this.counter += value
    }
}
```

Shared Mutable State

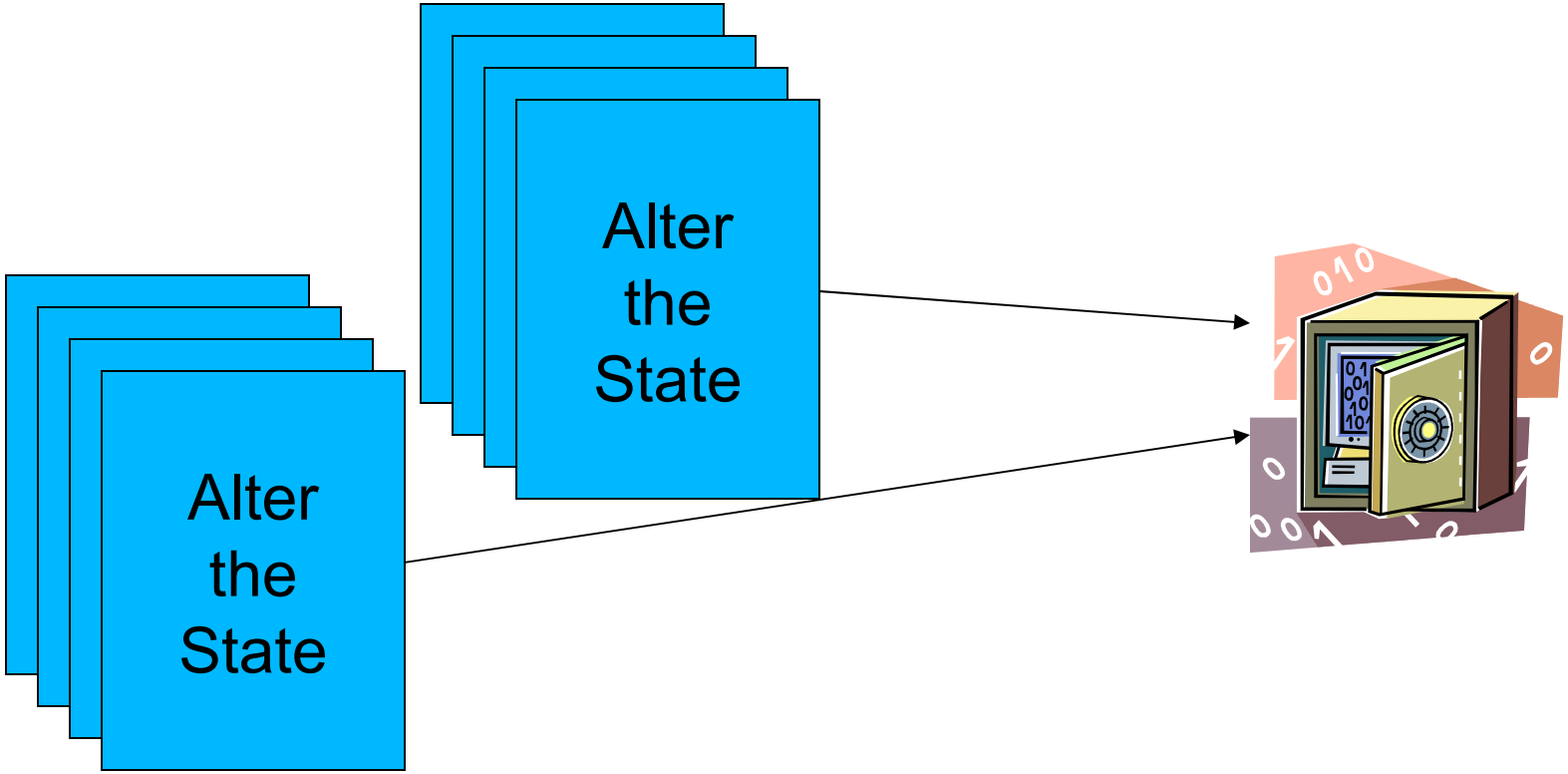
Misused most of the time

When really needed, use

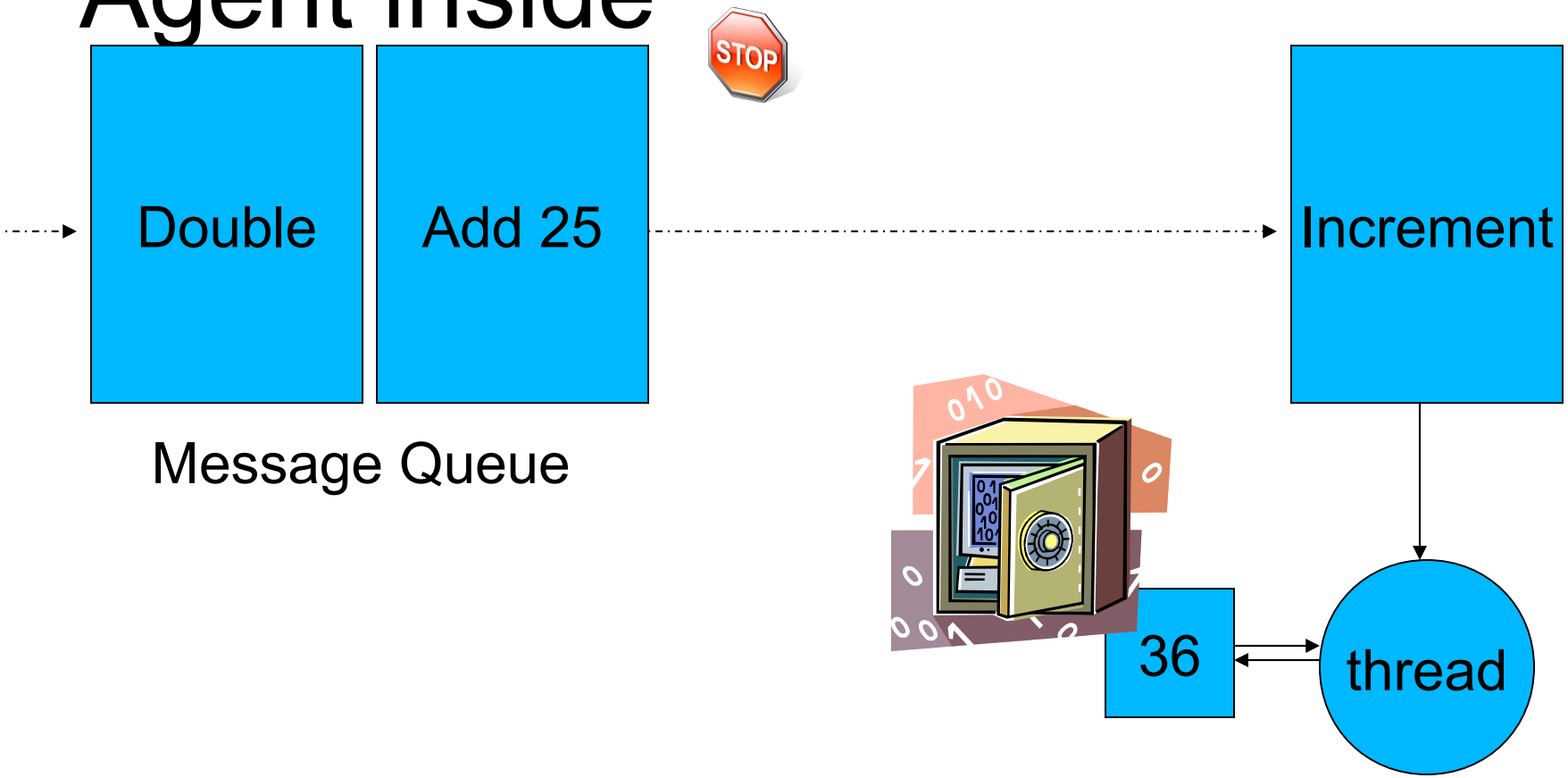
- Agents
- Software Transactional Memory
- Locks

Agent

- Lock Shared Mutable State in a Safe



Agent inside



Agent (ScalaAgent)

```
def increment(x: Long) = x + 1
def decrement(delta : Long)(x: Long) = x - delta
val agent = Agent(0L)

agent(increment _)
agent(decrement(3) _)
agent{_ + 100}

println(agent get)
```

STM (Akka - Scala)

```
atomic {
    .. // do something within a transaction
}

atomic(maxNrOfRetries) { .. }
atomicReadOnly { .. }

atomically {
    .. // try to do something
} orElse {
    .. // if tx clash; try do do something else
}
```

Sample taken from Akka documentation - http://doc.akkasource.org/stm

Persistent Data Structures

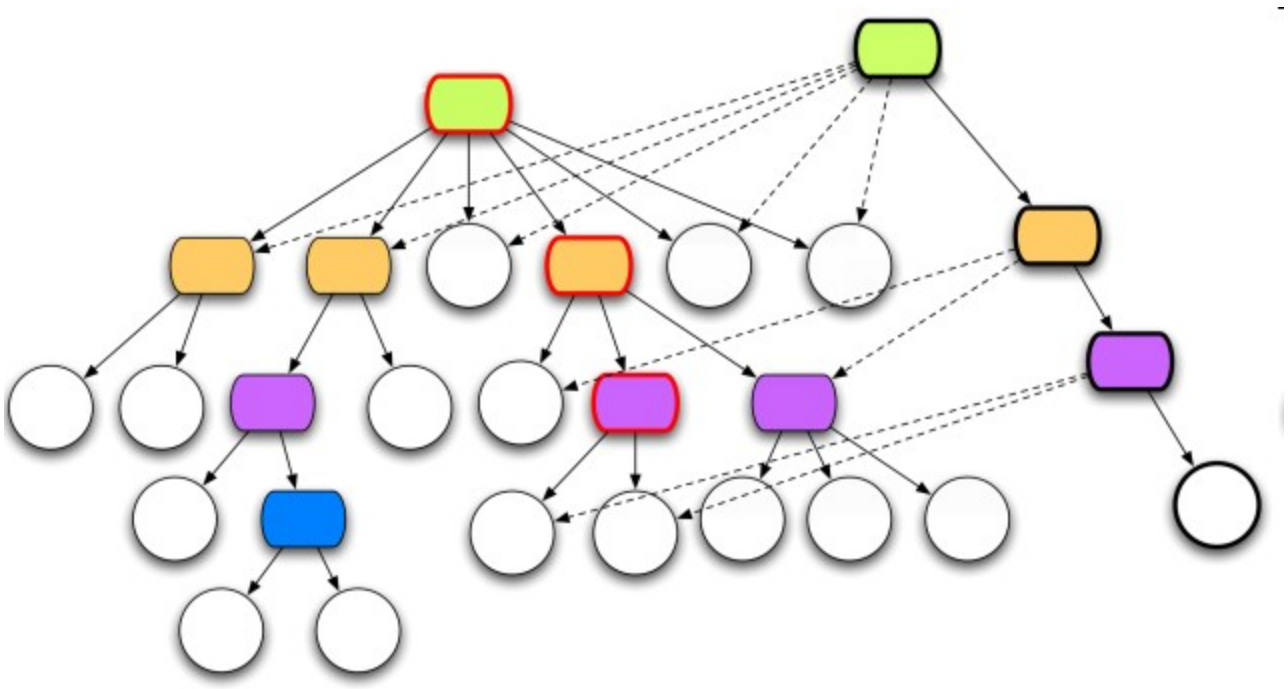


Illustration taken from Rich Hickey's presentation. Copyright Rich Hickey 2009

No more threads and locks

```
images.eachParallel {
    //concurrency agnostic code here
}

def myActor = actor {
    //concurrency agnostic code here
}

atomic { /*concurrency agnostic code here*/ }
…
```

Summary

Parallelism is not hard, multi-threading is

Jon Kerridge, Napier University

References

http://gpars.codehaus.org

http://akka.io

http://g.oswego.edu/dl/concurrency-interest/