The do's and don'ts of error handling

Joe Armstrong

A system is
fault tolerant
if it continues working
even if something is
wrong

Work like this is never finished
it’s always in-progress

- Hardware can fail
  - relatively uncommon

- Software can fail
  - common

Overview

- Fault-tolerance cannot be achieved using a single computer
  - it might fail

- We have to use several computers
  - concurrency
  - parallel programming
  - distributed programming
  - physics
  - engineering
  - **message passing is inevitable**

- Programming languages should make this ~~easy~~ doable

- How individual computers work is the smaller problem

- How the computers are interconnected and the protocols used between the computers is the significant problem

- We want the same way to program large and small scale systems

Message passing is inevitable

Message passing is the basis of OOP

prototypes vs classes was: Re: Sun's HotSpot

Alan Kay alank at wdi.disney.com
Sat Oct 10 04:40:35 UTC 1998

- Previous message: prototypes vs classes was: Re: Sun's HotSpot
- Next message: prototypes vs classes
- **Messages sorted by:** [date] [thread] [subject] [author]

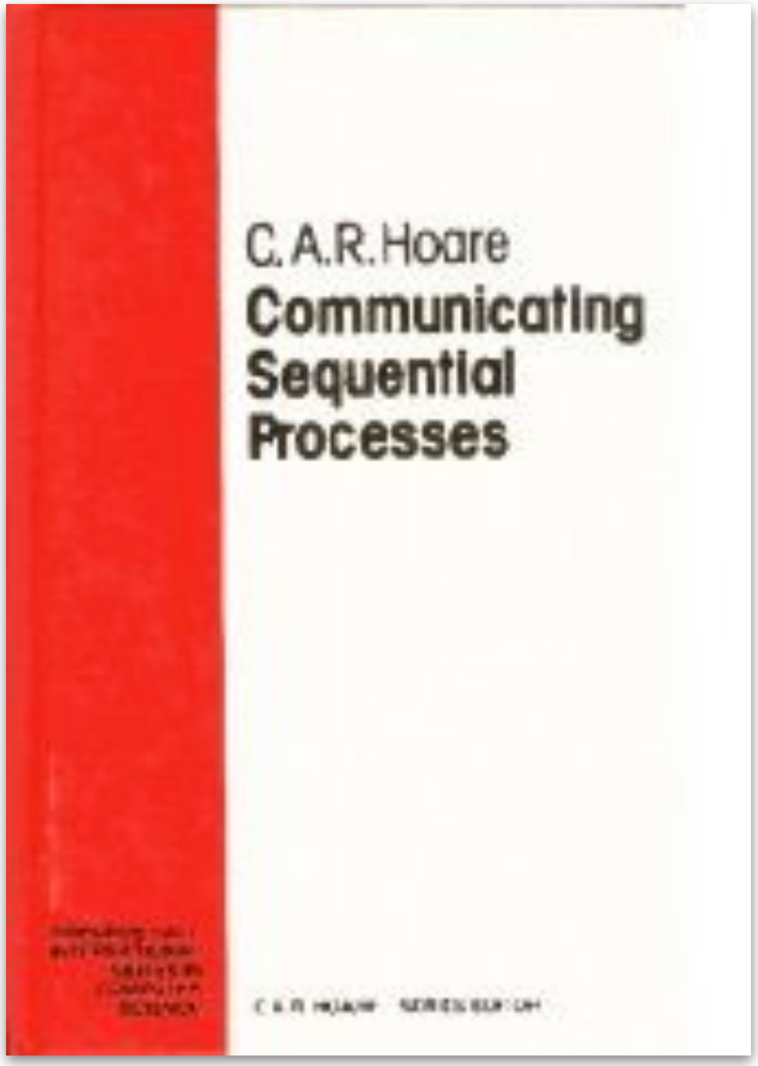
Folks --

Just a gentle reminder that I took some pains at the last OOPSLA to try to remind everyone that Smalltalk is not only NOT its syntax or the class library, it is not even about classes. I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea.

The big idea is "messaging" -- that is what the kernal of Smalltalk/Squeak is all about (and it's something that was never quite completed in our Xerox PARC phase). The Japanese have a small word -- ma -- for "that which is in between" -- perhaps the nearest English equivalent is "interstitial". The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be. Think of the internet -- to live, it (a) has to allow many different kinds of ideas and realizations that are beyond any single standard and (b) to allow varying degrees of safe interoperability between these ideas.

If you focus on just messaging -- and realize that a good metasystem can late bind the various 2nd level architectures used in objects -- then much of the language-, UI-, and OS based discussions on this thread are really quite moot. This was why I complained at the last OOPSLA that -- whereas at

And CSP



Erlang

- Derived from Smalltalk and Prolog (influenced by ideas from CSP)
- Unifies ideas on concurrent and functional programming
- Follows laws of physics (asynchronous messaging)
- Designed for programming fault-tolerant systems

Building fault-tolerant software boils down to detecting errors and doing something when errors are detected

Types of errors

- Errors that can be detected at compile time
- Errors that can be detected at run-time
- Errors that can be inferred
- Reproducible errors
- Non-reproducible errors

Philosophy

- Find methods to prove SW correct at compile-time
- Assume software is incorrect and will fail at run time then do something about it at run-time

Evidence for
SW failure is
all around us

Proving the self-consistency of small programs will not help

Why self-consistency?

Proving things is difficult

- Prove the Collatz conjecture (also known as the Ulam conjecture, Kakutani's prolem, Thwaites conjecture, Hasse's algorithm or the Syracuse problem)

3N+1

- If N is odd replace it by 3N+1

- If N is even replace it by N/2

The Collatz conjecture is:
This process will eventually reach the number 1, for all starting values on N

"Mathematics may not be ready for such problems"

Paul Erdős

Conclusion

- Some small things can be proved to be self-consistent
- Large assemblies of small things are impossible to prove correct

Timeline

Erlang model of computation rejected. Shared memory systems rule the world

- 1980 - Rymdbolaget - first interest in Fault-tolerance - Viking Satellite
- 1985 - Ericsson - start working on “a replacement PLEX” - start thinking about errors - “errors must be corrected somewhere else” “shared memory is evil” “pure message passing”
- 1986 - Erlang - unification of OO with FP
- 1998 - Several products in Erlang - Erlang is banned
- 1998 .. 2002 - Bluetail -> Alteon -> Nortel -> Fired
- 2002 - I move to SICS
- 2003 - Thesis
- 2004 - Back to Ericsson
- 2015 - Put out to grass

Erlang model of computation widely accepted and adopted in many different languages

Viking

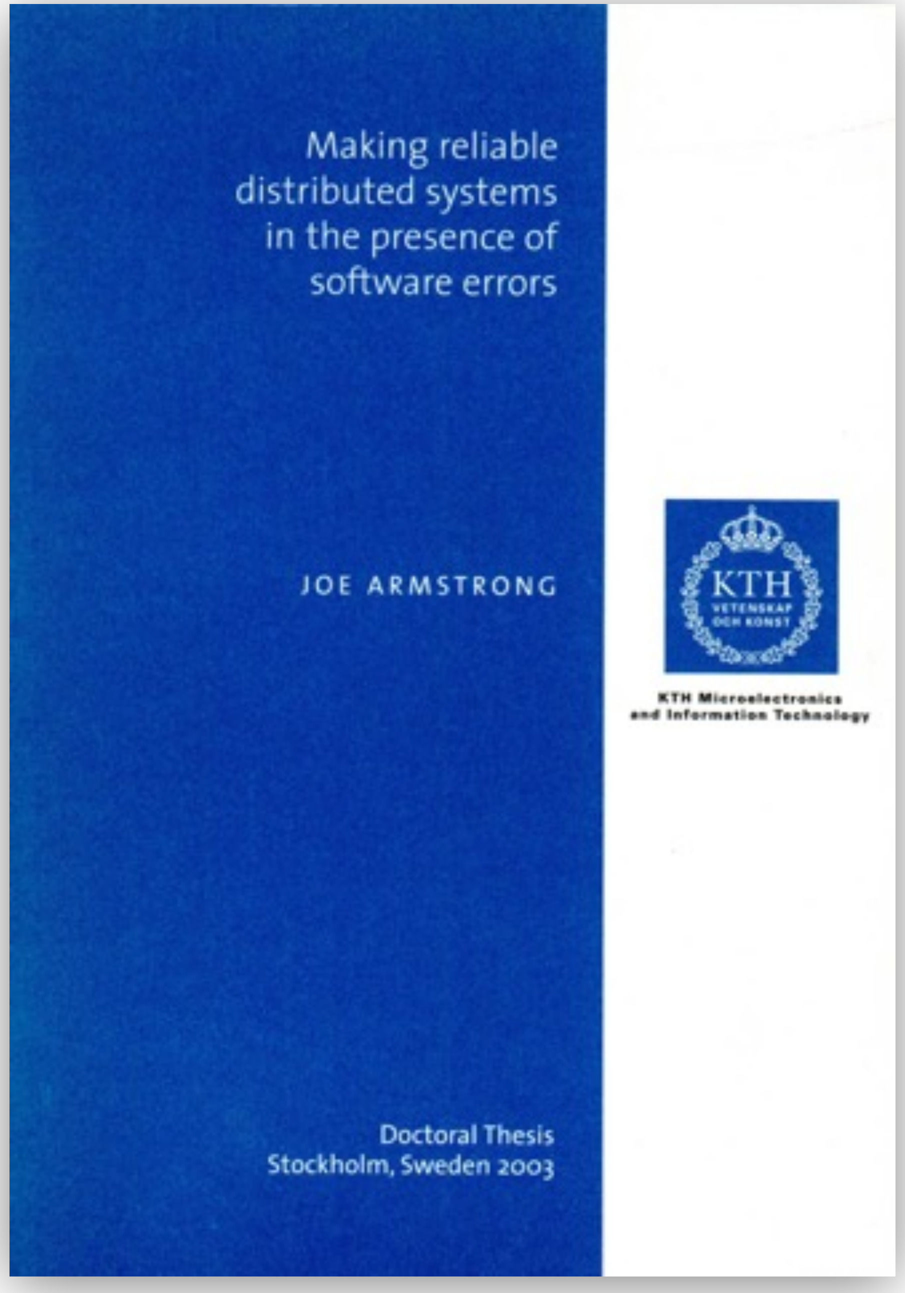
Incorrect
Software
is not an option

Types of system

- Highly reliable (nuclear power plant control, air-traffic) - satellite (very expensive if they fail)
- Reliable (driverless cars) (moderately expensive if they fail. Kills people if they fail)
- Reliable (Annoys people if they fail) banks, telephone
- Dodgy - (Cross if they fail) Internet - HBO, Netflix
- Crap - (Very Cross if they fail) Free Apps

Different technologies are used to build and validate the systems

How can we
make software that
works reasonably well
even if there are
errors in the software?



http://erlang.org/download/
armstrong_thesis_2003.pdf

Requirements

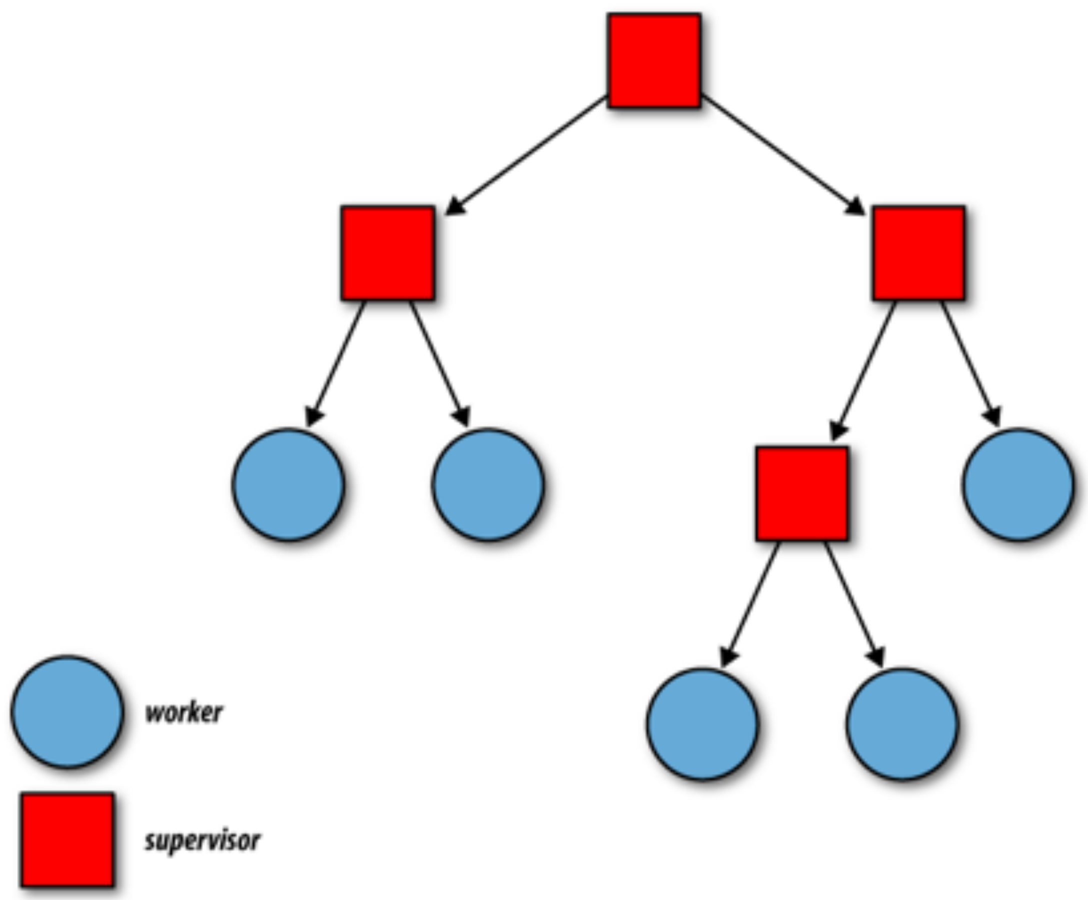
- R1 - Concurrency
- R2 - Error encapsulation
- R3 - Fault detection
- R4 - Fault identification
- R5 - Code upgrade
- R6 - Stable storage

Source: Armstrong thesis 2003

The “method”

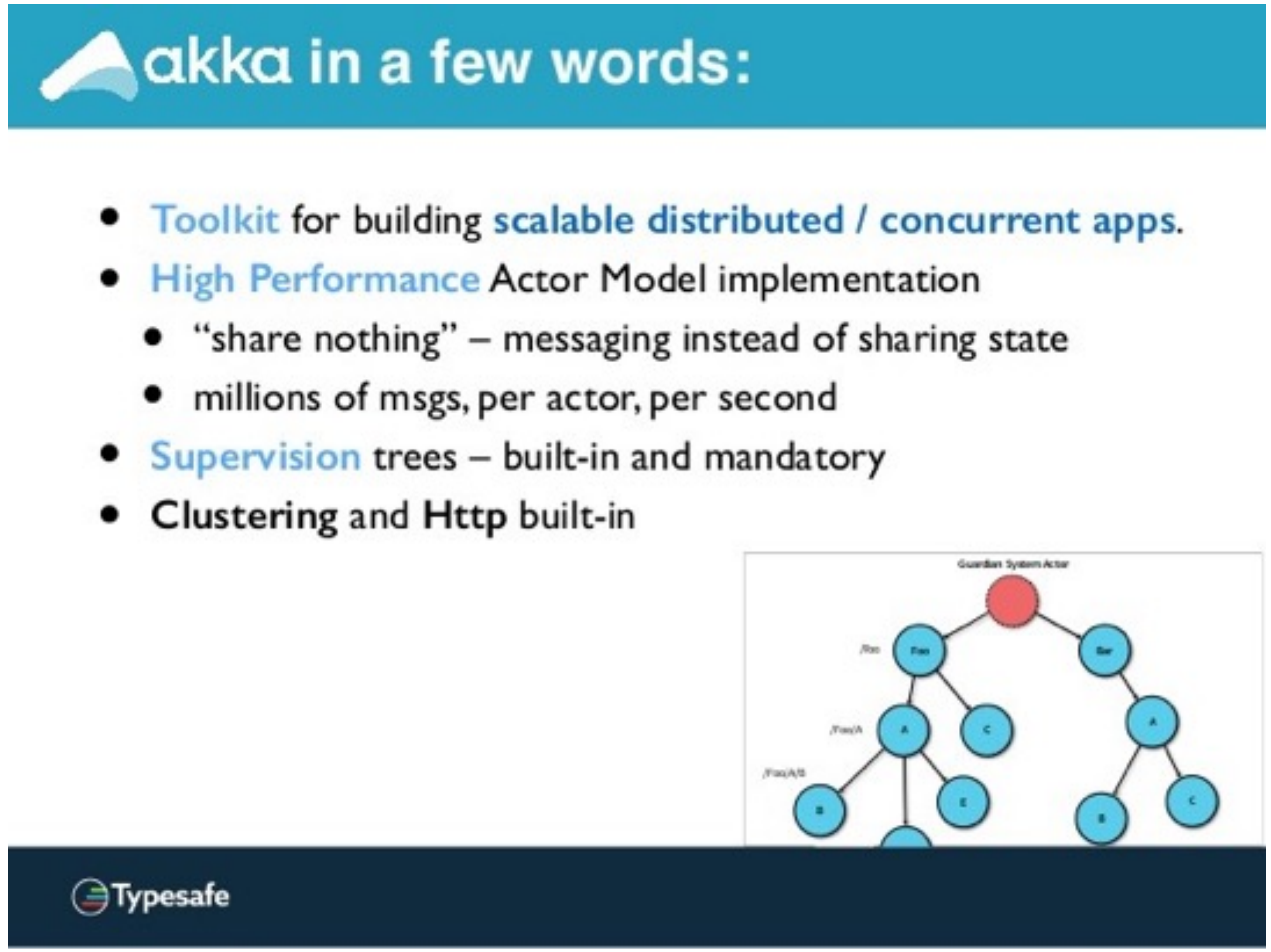
- Detect all errors (and crash???)
- If you can’t do what you want to do try to do something simpler
- Handle errors “remotely” (detect errors and ensure that the system is put into a safe state defined by an invariant)
- Identify the “Error kernel”
 (the part that must be correct)

Supervision trees

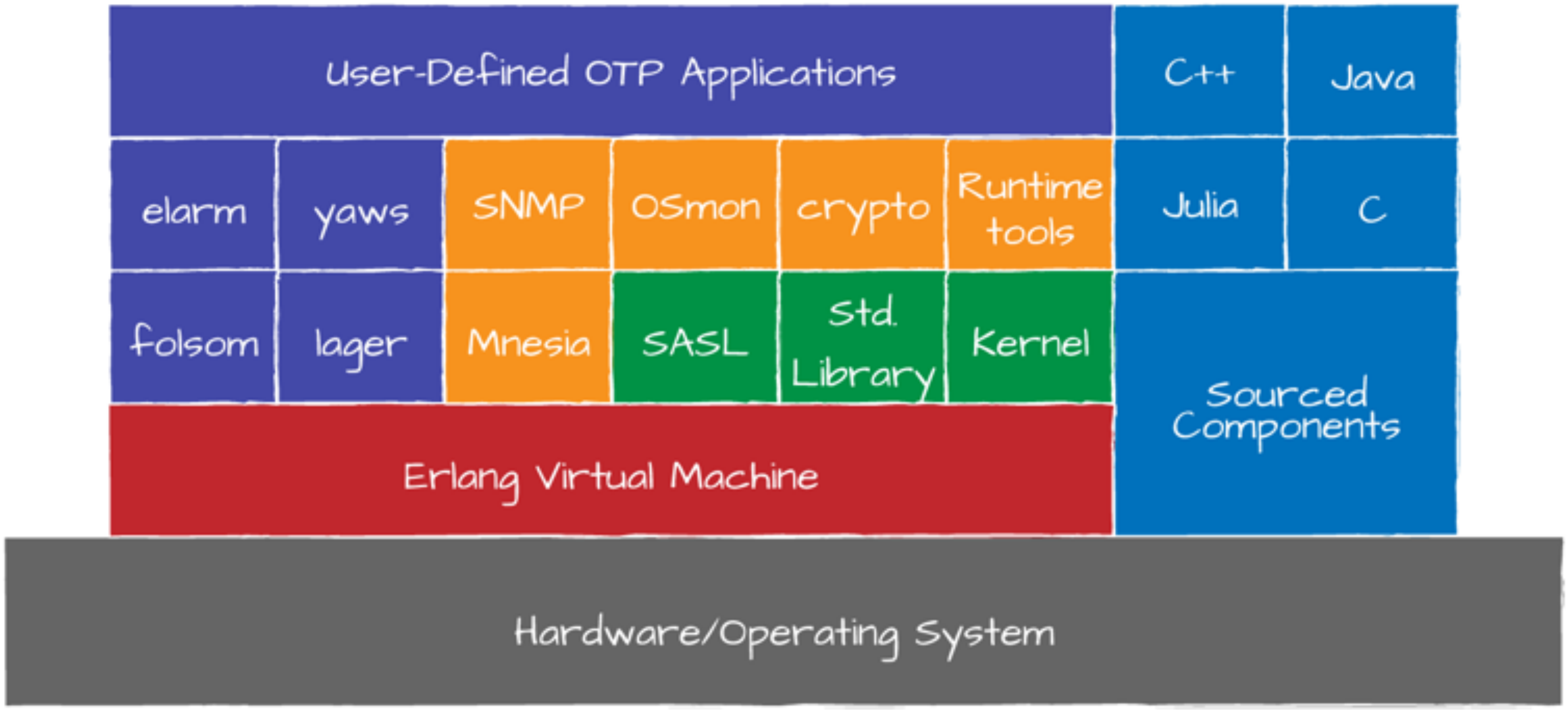


Note: nodes
can be on different
machine

From: Erlang Programming
Cesarini & Thompson 2009



Akka is "Erlang supervision for Java and Scala"



Source: Designing for Scalability with Erlang/OTP
Cesarini & Vinoski O'Reilly 2016

It works

- Ericsson smart phone data setup
- WhatsApp
- CouchDB (CERN - *we found the higgs*)
- Cisco (netconf)
- Spine2 (NHS - uk - riak (basho) replaces Oracle)
- RabbitMQ

- What is an error ?
- How do we discover an error ?
- What to do when we hit an error ?

What is an error?

- An undesirable property of a program
- Something that crashes a program
- A deviation between desired and observed behaviour

Who finds the error?

- The program (run-time) finds the error
- The programmer finds the error
- The compiler finds the error

The run-time finds an error

- Arithmetic errors
  divide by zero, overflow, underflow, …
- Array bounds violated
- System routine called with nonsense arguments
- Null pointer
- Switch option not provisioned
- An incorrect value is observed

What should the run-time do when it finds an error?

- Ignore it (no)
- Try to fix it (no)
- Crash immediately (yes)

- Don't Make matters worse
- Assume somebody else will fix the problem

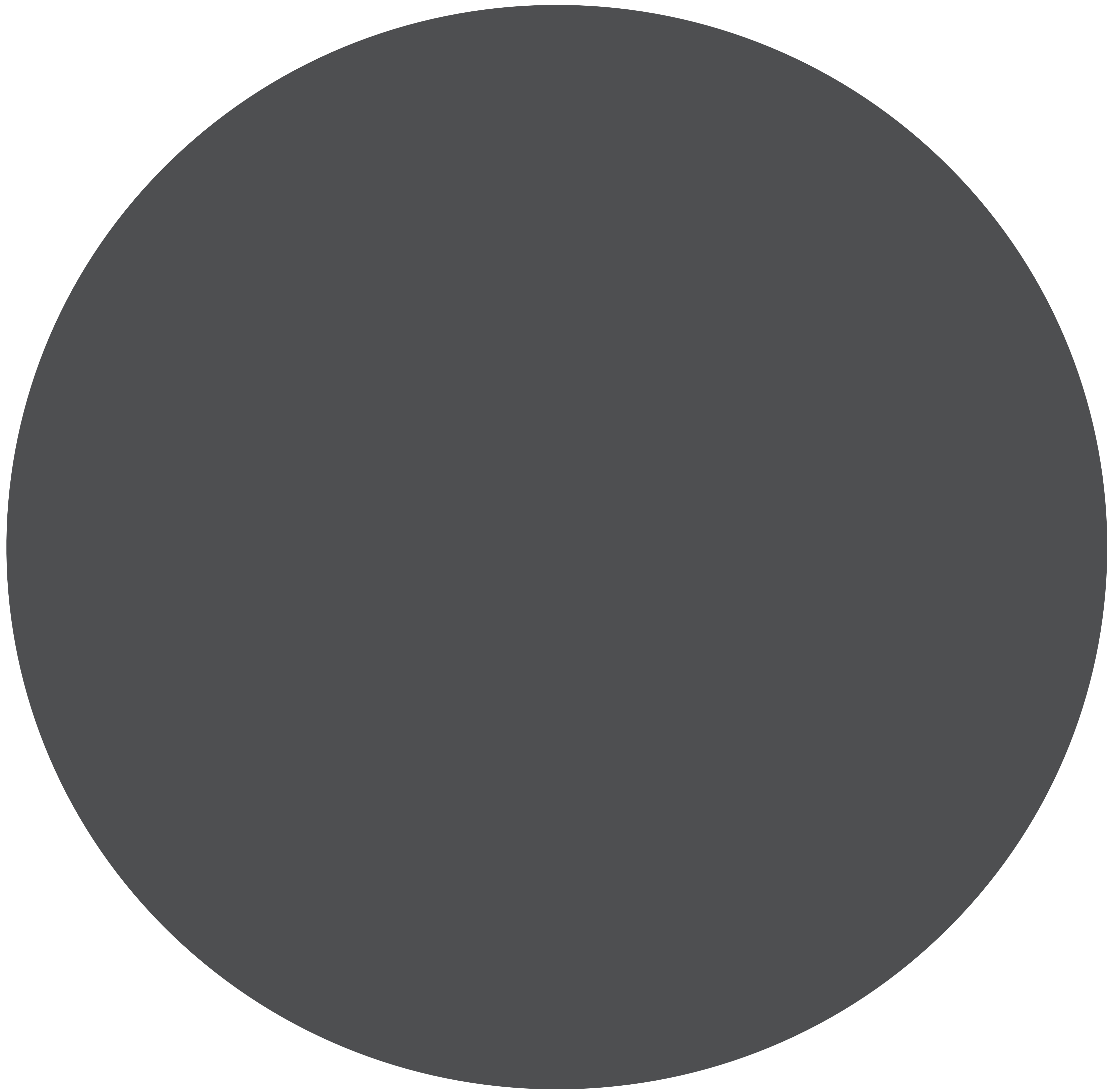
What should the programmer do when they don’t know what to do?

- Ignore it (no)
- Log it (yes)
- Try to fix it (possibly, but don’t make matters worse)
- Crash immediately (yes)

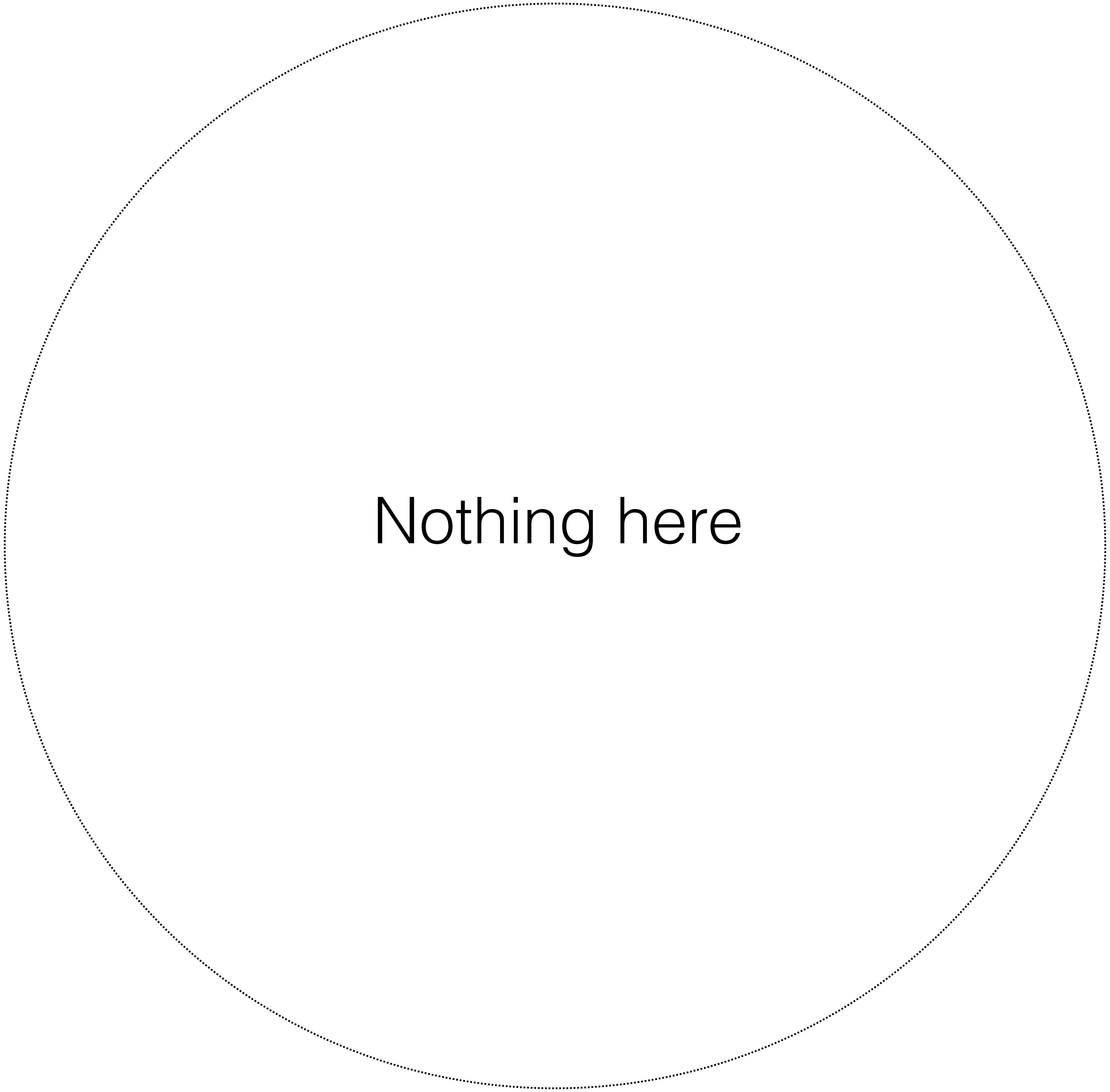
In sequential languages with single threads crashing is not widely practised

What’s the big deal about concurrency?

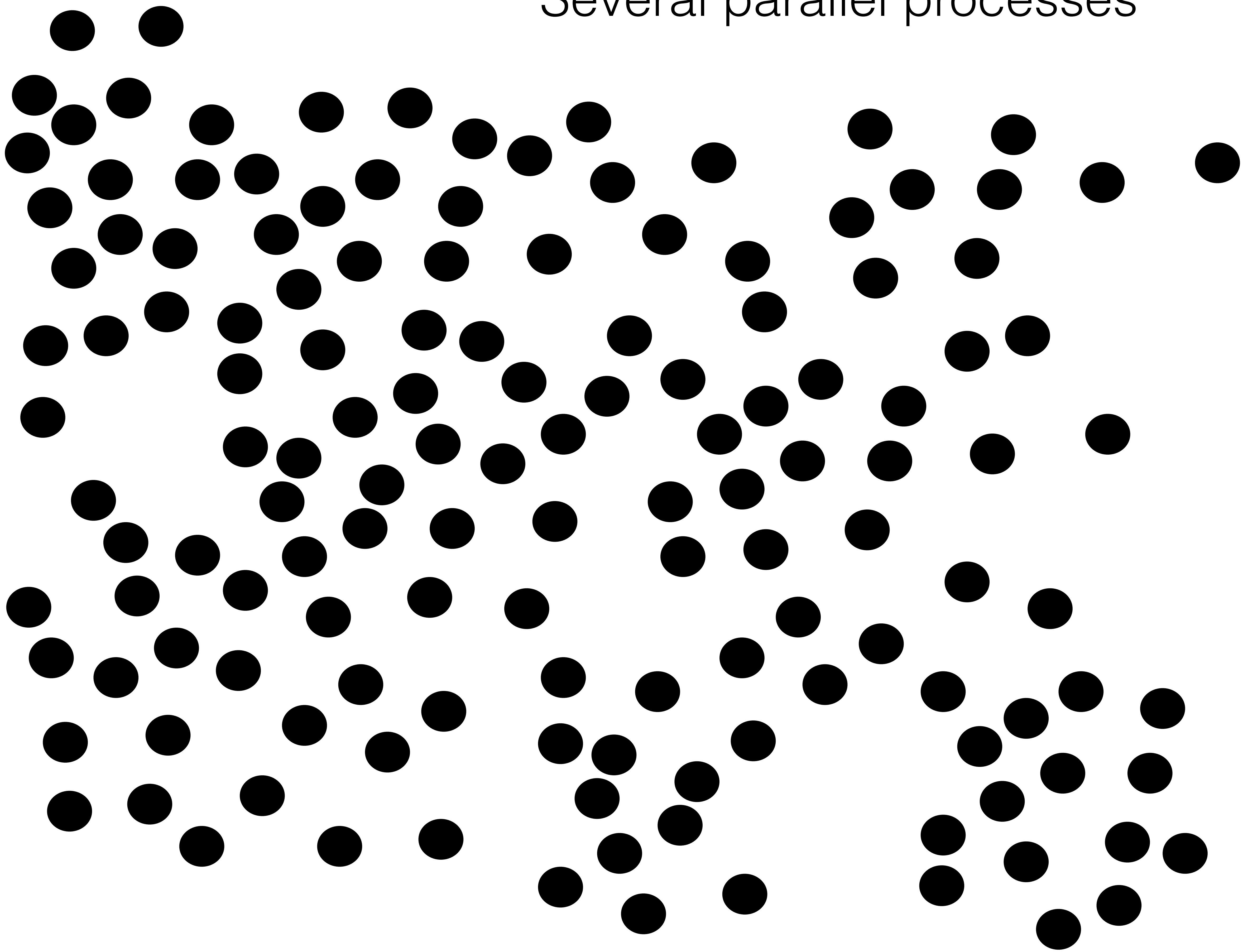
A sequential program



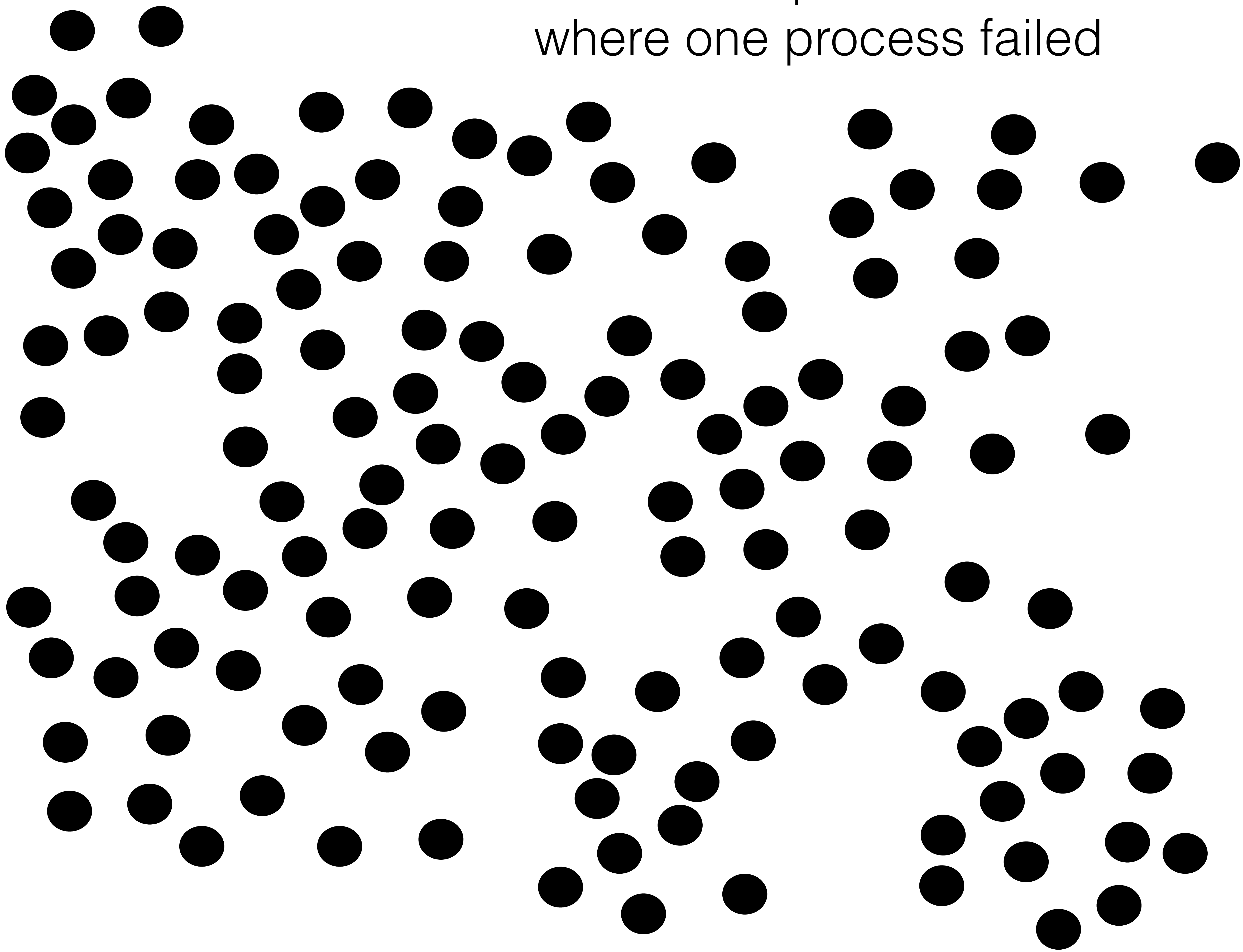
A dead sequential program



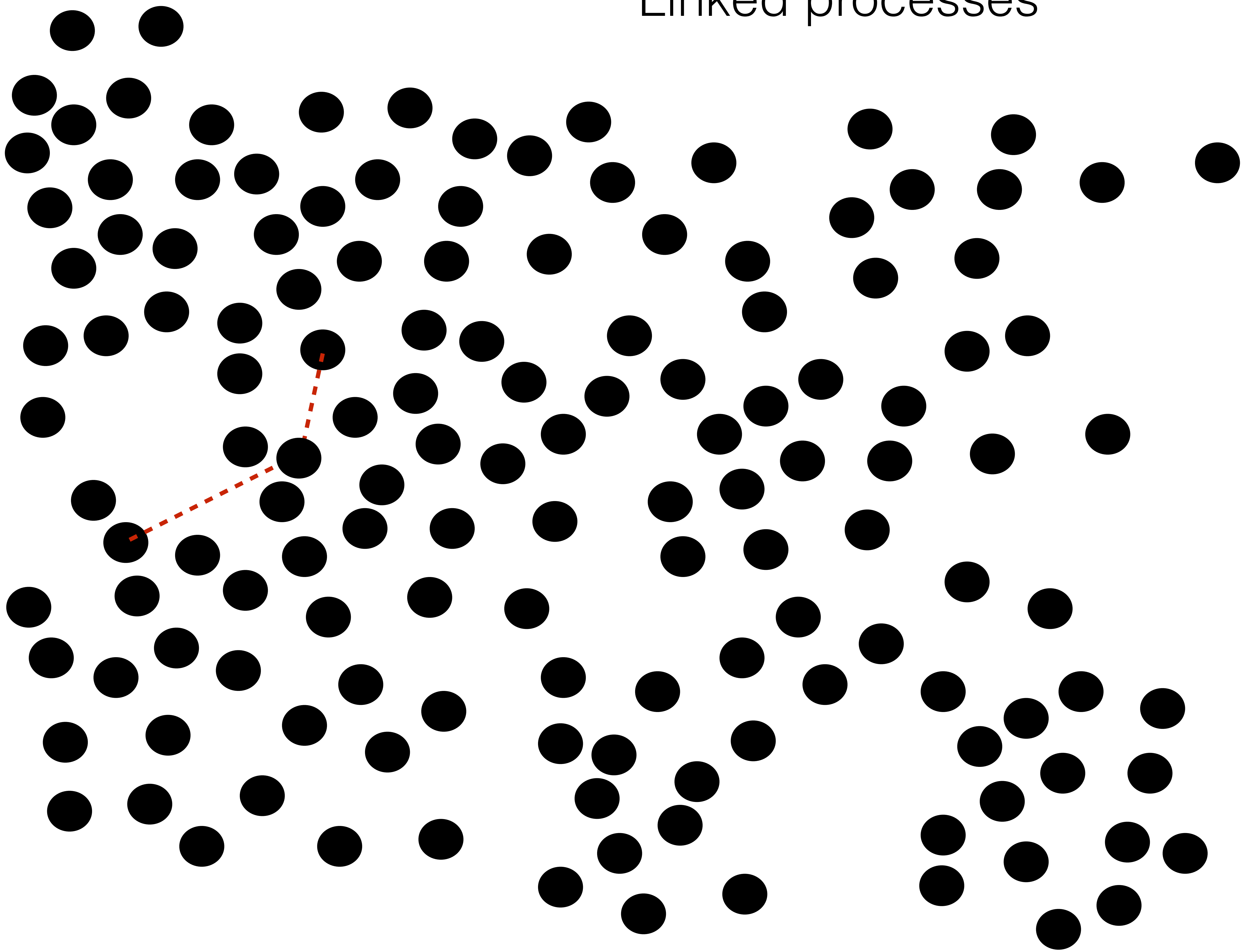
Several parallel processes



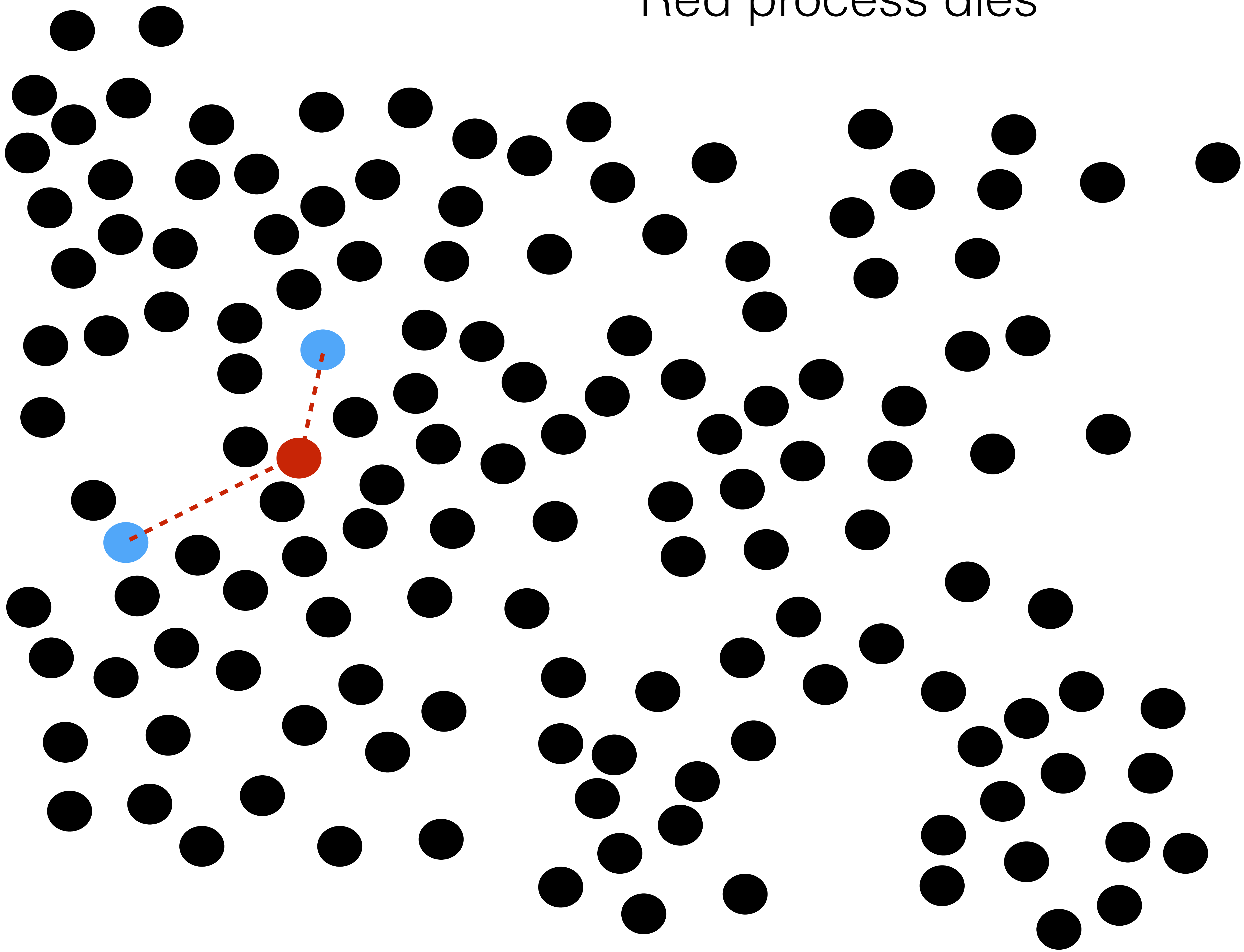
Several processes
where one process failed



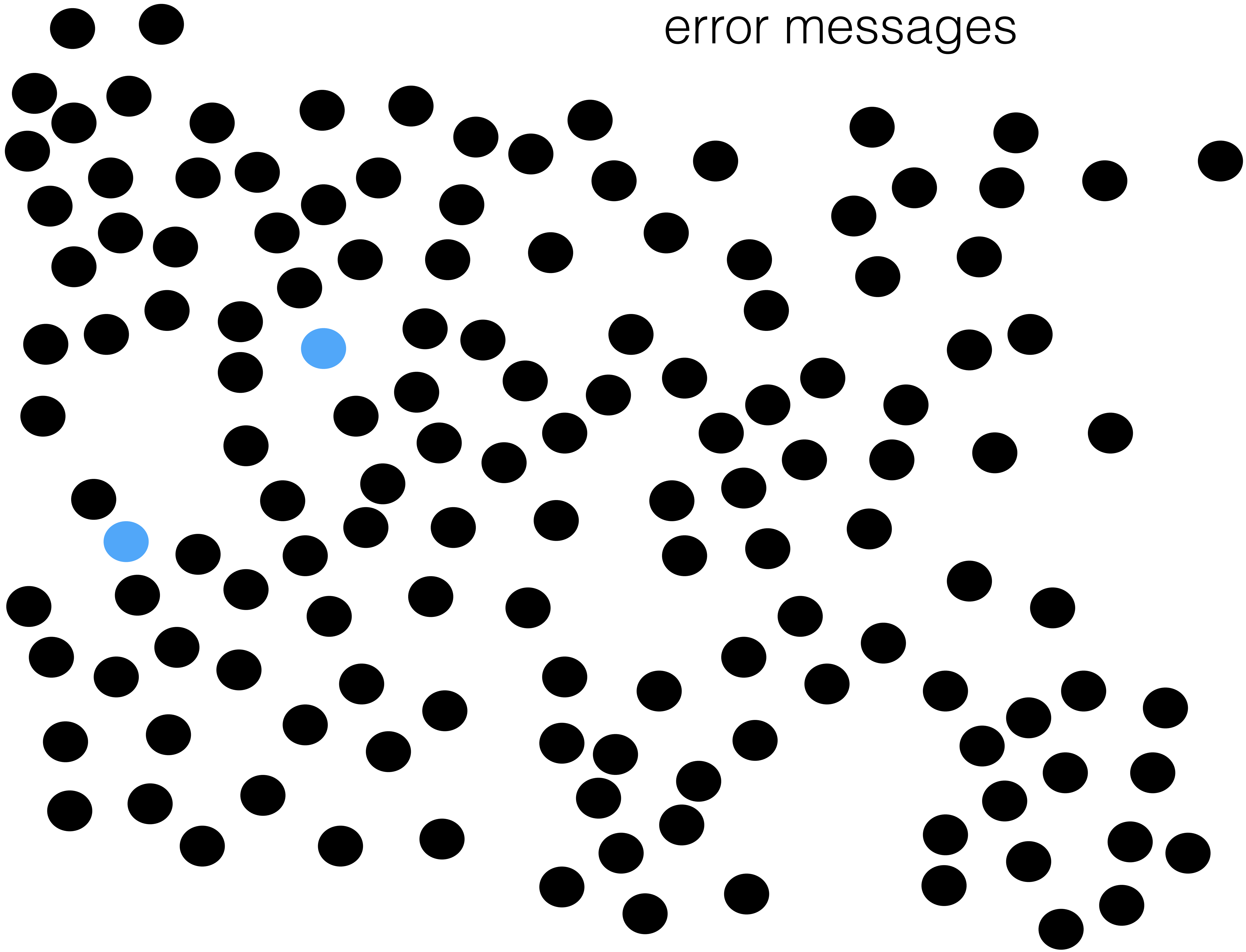
Linked processes



Red process dies



Blue processes are sent error messages



Why concurrent?

Fault-tolerance
is impossible
with one computer

AND

Scalable is
impossible
with one computer *

* To more than the capacity of
the computer

AND

Security is very difficult with one computer

AND

I want one way to program
not two ways
one for local systems
the other for distributed systems
(rules out shared memory)

Detecting Errors

Where do errors come from?

- Arithmetic errors
- Unexpected inputs
- Wrong values
- Wrong assumptions about the environment
- Sequencing errors
- Concurrency errors
- Breaking laws of maths or physics

Arithmetic Errors

- *silent **and deadly** errors* - errors where the program does not crash but delivers an incorrect result

- *noisy errors* - errors which cause the program to crash

Silent Errors

- “quiet” NaN’s
- arithmetic errors

- these make matters worse

mie verhoogd. Uw premie was € NaN per maand en wordt € 13,56 p
ook op de bijgevoegde polis. Als u het niet eens bent met deze aa
ëindigen.

A nasty silent error

Oops?

http://www.military.com/video/space-technology/launch-

```
    end if;
    L_M_DON_32 := TDB.T_ENTIER_32S ((1.0/C_M_LSB_DON) *
                                   G_M_INFO_DERIVE(T_ALG.E_DON))
    if L_M_DON_32 > 32767 then
       P_M_DERIVE(T_ALG.E_DON) := 16#7FFF#;
    elsif L_M_DON_32 < -32768 then
       P_M_DERIVE(T_ALG.E_DON) := 16#8000#;
    else
       P_M_DERIVE(T_ALG.E_DON) := UC_16S_EN_16NS(
          TDB.T_ENTIER_16S(L_M_DON_32));
    end if;

    P_M_DERIVE(T_ALG.E_DOE) := UC_16S_EN_16NS (TDB.T_ENTIER_16S
                                    ((1.0/C_M_LSB_DOE) *
                                    G_M_INFO_DERIVE(T_ALG.E_DOE)

    L_M_BV_32 := TDB.T_ENTIER_32S ((1.0/C_M_LSB_BV) *
                                   G_M_INFO_DERIVE(T_ALG.E_BV));
    if L_M_BV_32 > 32767 then
       P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
    elsif L_M_BV_32 < -32768 then
       P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
    else
       P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M
    end if;

    P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS (TDB.T_ENTIER_16S
                                   ((1.0/C_M_LSB_BH) *
                                   G_M_INFO_DERIVE(T_ALG.E_BH)))
  end LIRE_DERIVE;
--$finprocedure

  --(
  procedure LIRE_SEUIL (P_M_SEUIL : out TDB.T_ENTIER_16NS) is
    --)
```

Juin 1996

501

http://moscova.inria.fr/~levy/talks/10enslongo/enslongo.pdf

Silent Programming Errors

Why silent? because the programmer does not know there is an error

Rump's Royal Pain

Compute $333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$
where $x = 77617$, $y = 33096$.

- Using IBM (pre-IEEE Standard) floats, Rump got
  - 1.172603 in 32-bit precision
  - 1.1726039400531 in 64-bit precision
  - 1.172603940053178 in 128-bit precision
- Using IEEE double precision: 1.18059×10^{21}
- **Correct answer: –0.827396059946821368…!**

Didn't even get *sign* right

The end of numerical Error
John L. Gustafson, Ph.D.

Beyond Floating Point:
Next generation computer arithmetic
John Gustafson

(Stanford lecture)

https://www.youtube.com/watch?v=aP0Y1uAA-2Y

Arithmetic
is very difficult
to get right

- Same answer in single and double precision does not mean the answer is right

- If it matters you must prove every line containing arithmetic is correct

- Real arithmetic is not associative

Most programmers think
that a+(b+c) is the same as (a+b)+c

```
> ghci
Prelude> a = 0.1 + (0.2 + 0.3)
Prelude> a
0.6
Prelude> b = (0.1 + 0.2) + 0.3
Prelude> b
0.6000000000000001
Prelude> a == b
False
```

```
$ python
Python 2.7.10
>>> x = (0.1 + 0.2) + 0.3
>>> y = 0.1 + (0.2 + 0.3)
>>> x==y
False
>>> print('%.17f' %x )
0.60000000000000009
>>> print('%.17f' %y)
0.59999999999999998
```

```
$ erl
Eshell V9.0  (abort with ^G)
1> X = (0.1+0.2) + 0.3.
0.6000000000000001
2> Y = 0.1+ (0.2 + 0.3).
0.6
3> X == Y.
false
```

Most programming languages think
that a+(b+c) differs from (a+b)+c

Value errors

- Program does not crash, but the values computed are incorrect or inaccurate
- How do we know if a program/value is incorrect if we do not have a specification?
- Many programs have no specifications or specs that are so imprecise as to be useless
- The specification might be incorrect
  and the tests and the program

000042000021076035600
EXPEDITED PARCEL
COLIS ACCÉLÉRÉS
2
CANADA POST / POSTES CANADA
From / Exp.:
$retAdd.getFirstName().toUpperCase()
$retAdd.getAddressLine1().toUpperCase()
$retAdd.getCity().toUpperCase() $retAdd.getState().toUpperCase()
$retAdd.getDayPhone()
Payer / Facturé à:
7307904
Method of Payment /
Mode de paiement:
o / Dest.:

Programmer
does not know
what to do

CRASH

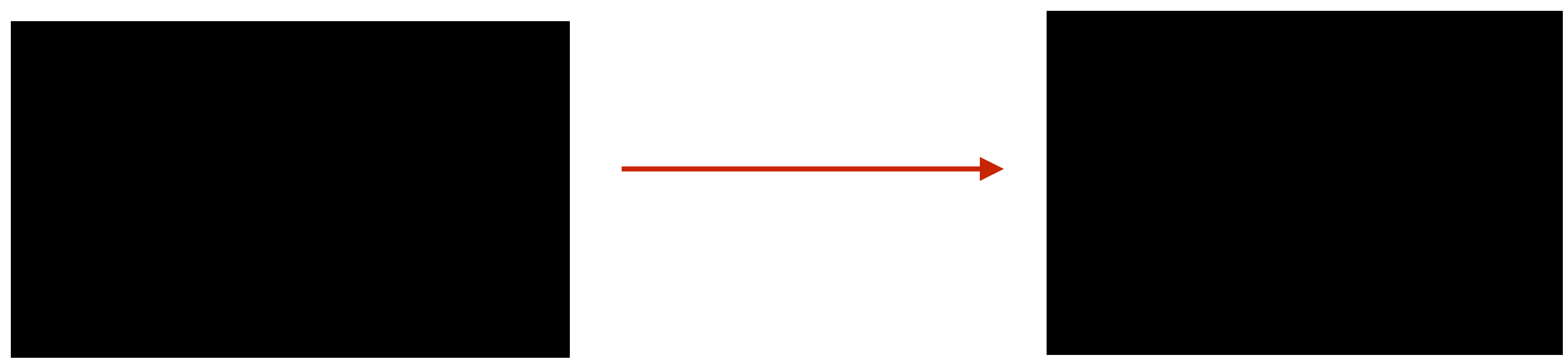
- *I call this "let it crash"*
- *Somebody else will fix the error*
- *Needs concurrency and links*

What do you do when you receive an error?

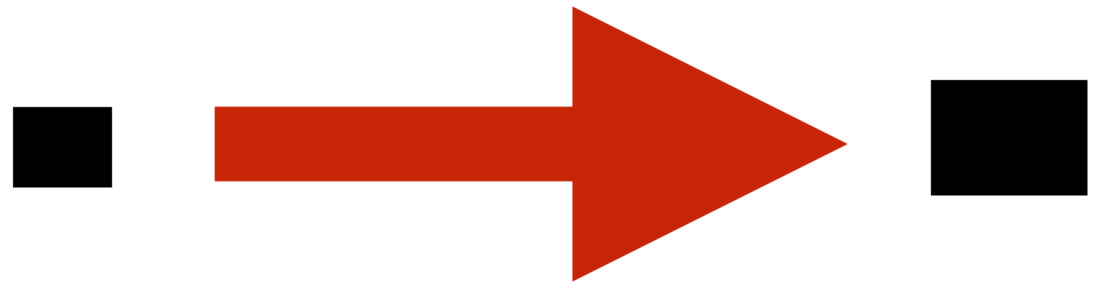
- Maintain an invariant
- Try to do something simpler

is that all?

What's in a message?



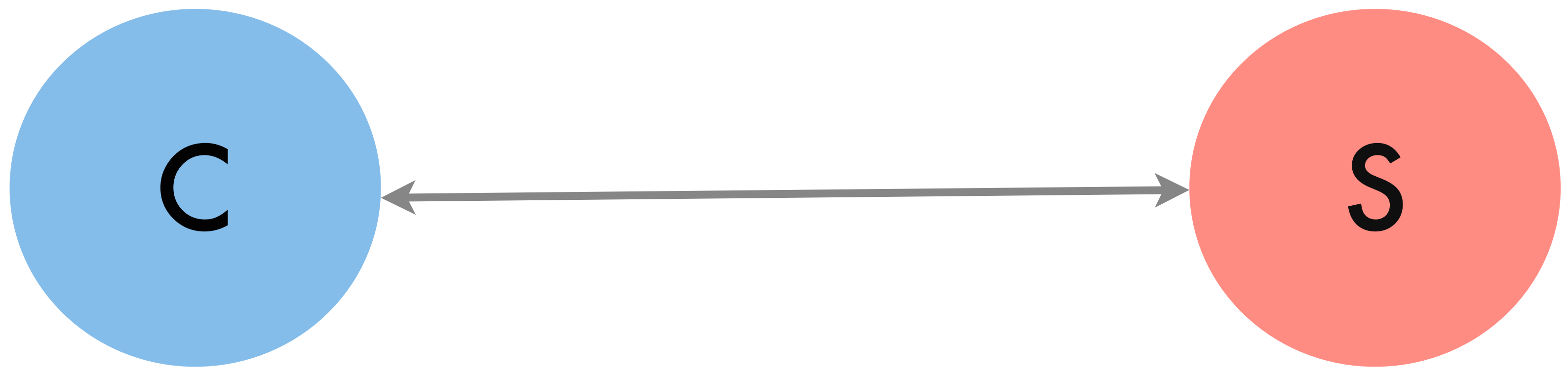
- Inside black boxes are programs
- There are thousands of programming languages
- What language used is irrelevant
- The only important thing is what happens at the interface
- Two systems are the same if they obey observational equivalence



- Interaction between components involves message passing
- There are very few ways to describe messages (JSON, XML)
- There are very very few formal ways to describe the valid sequences of messages (= protocols) between components (ASN.1)
  session types

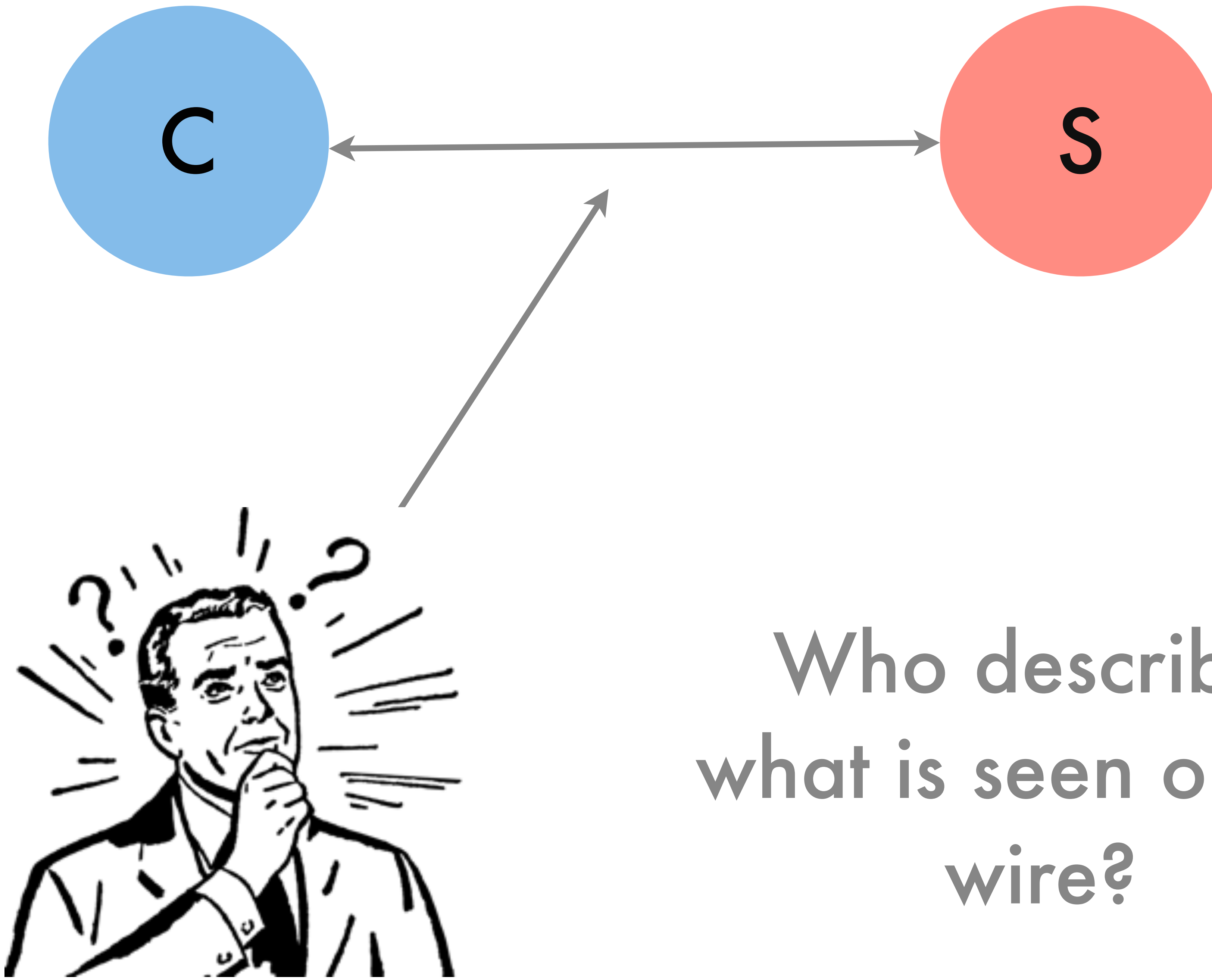
Protocols are contracts

Contracts assign blame



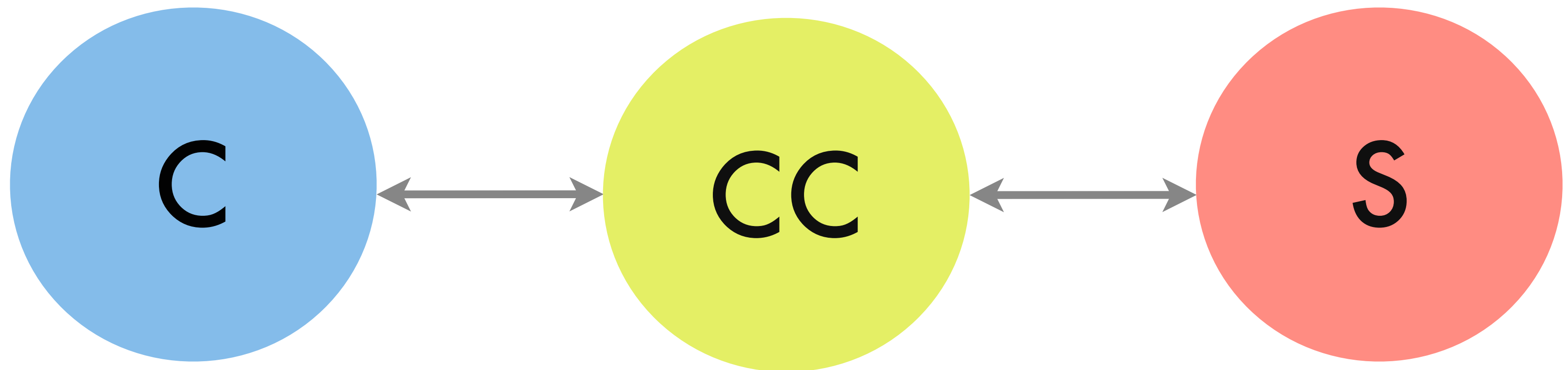
The client and server are isolated by a socket - so it should *"in principle"* be easy to change either the client or server, without changing the other side

But it's not easy

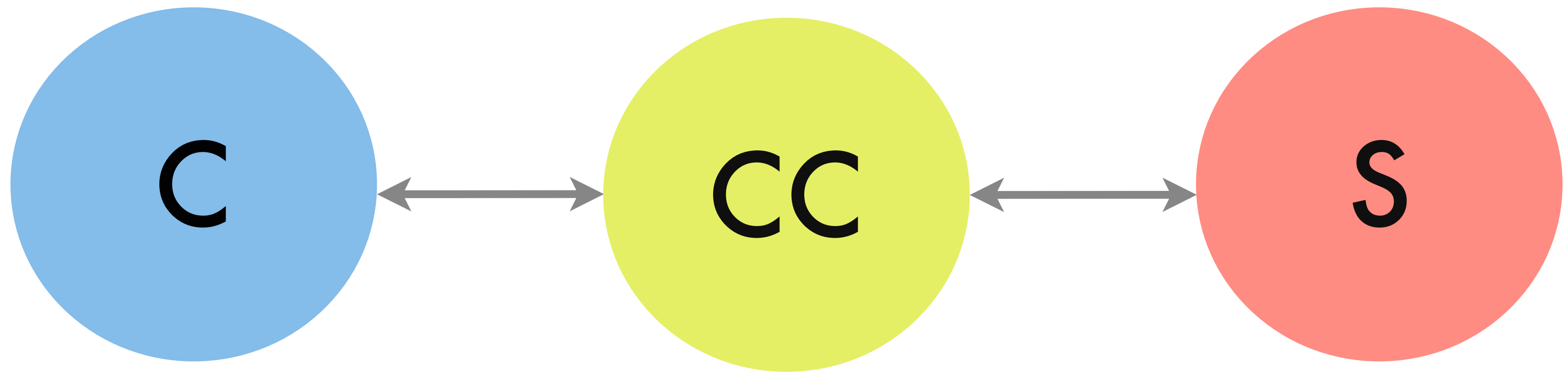


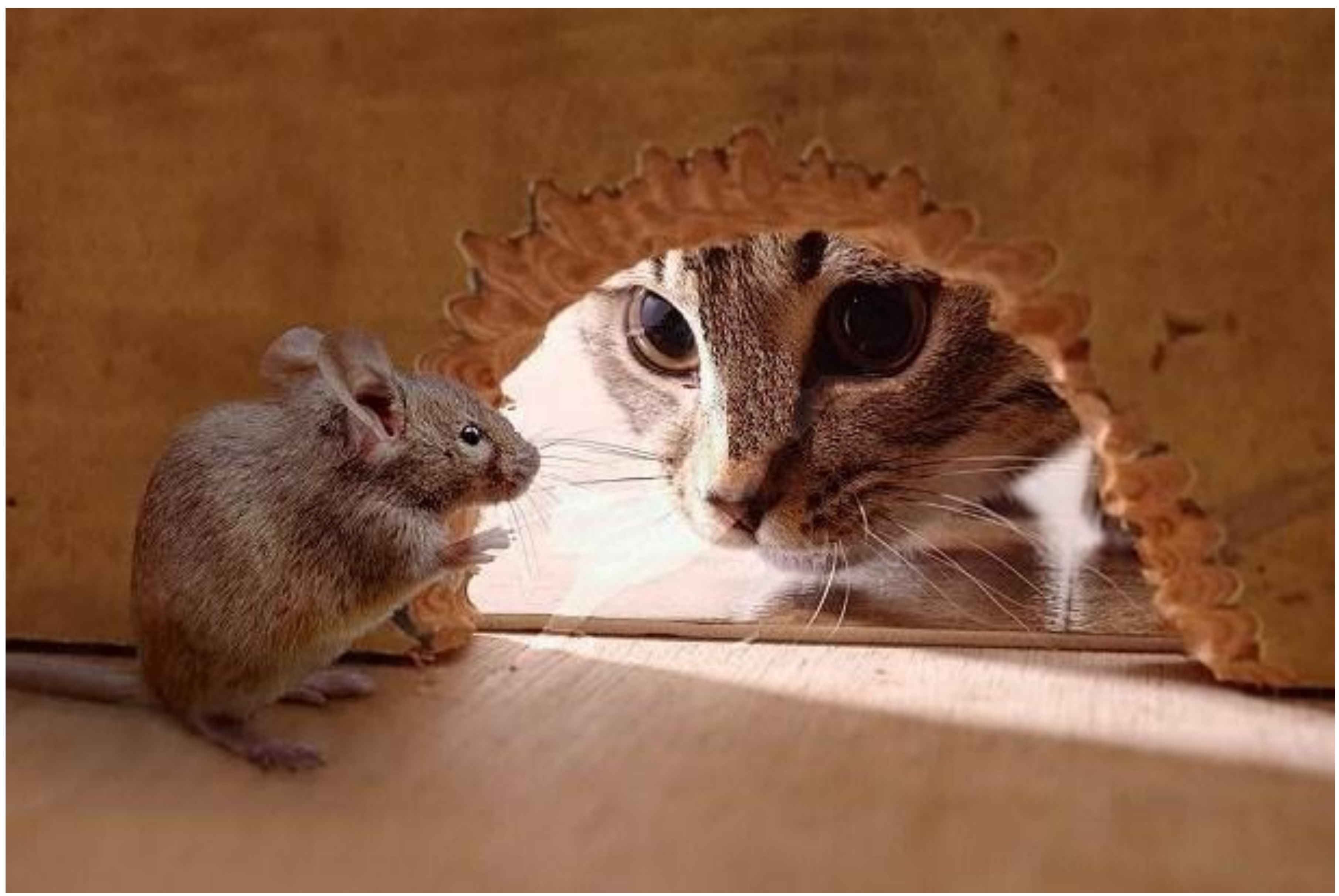
Who describes what is seen on the wire?


CONTRACT
THIS AGREEMENT made this
by
and between
and
day of
, 20
WITNESSETH: That in consideration
kept and performed on the
I. Said party of
(First Party)
(Second Party),
ovenants and agreements to be
ereto, respectively as herein stated:
ants and agrees that it shall:
d said party of the secon



The contract checker describes what is seen on the wire.


C
CC
S



How do we describe contracts?