# How to write fault-tolerant software

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

# 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
- 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)

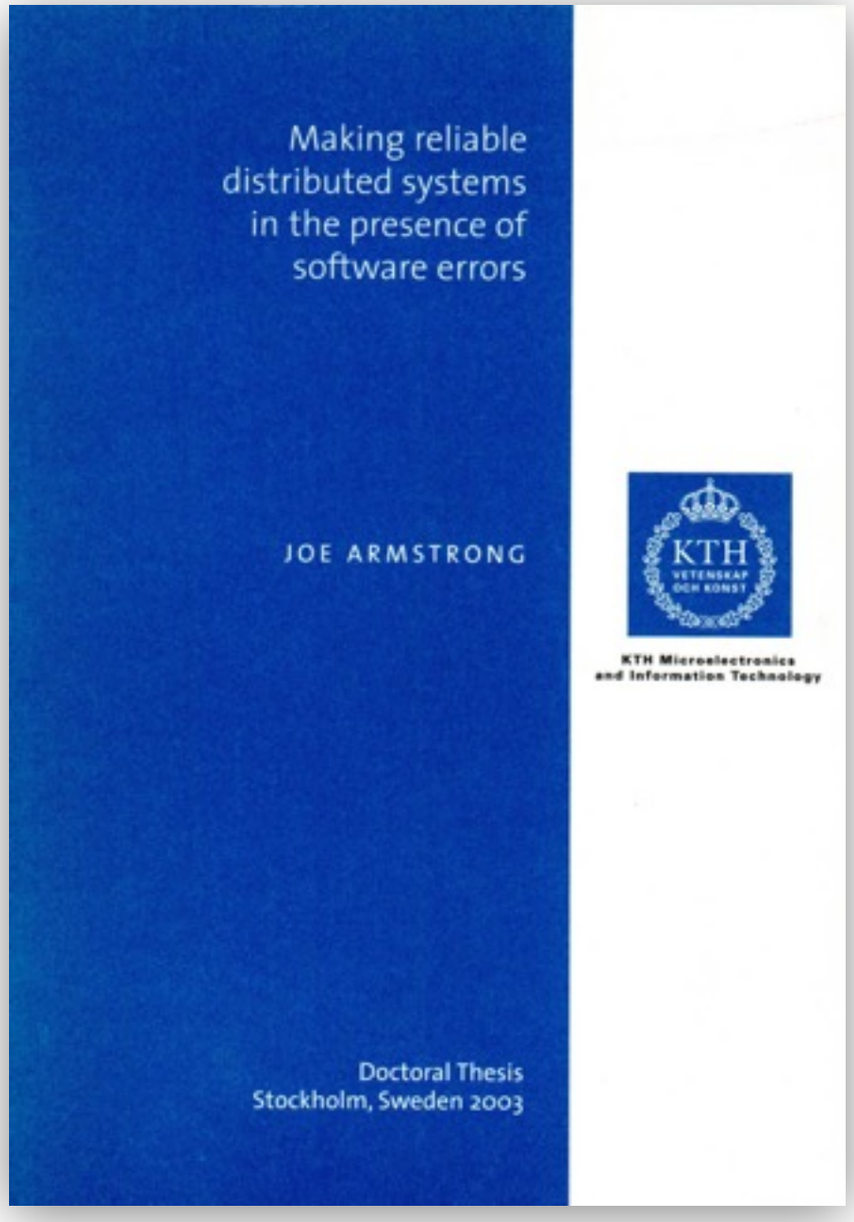
- Reliable (driverless cars) (moderately expensive if they fail. Kills people if they fail

- Reliable (Annoys people if they fail) banks, telephone

- Dodgy - (Very cross if they fail) Internet - HBO, Netflix

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

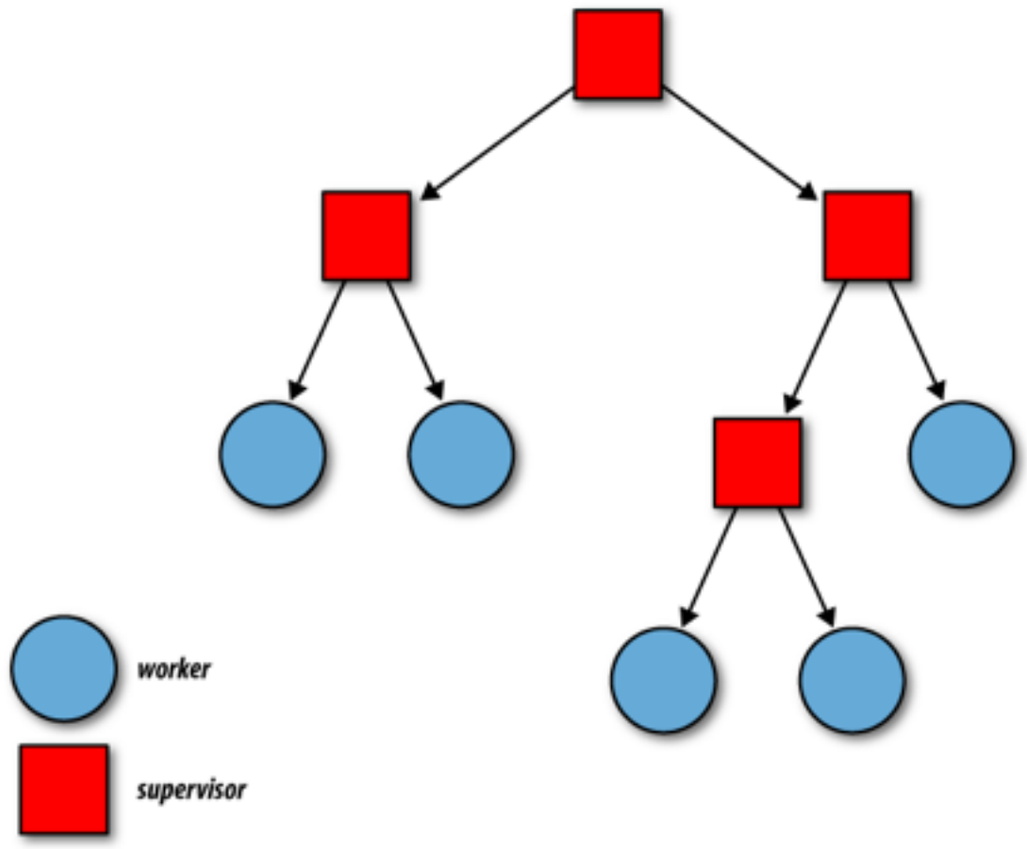
- R1 - Concurrency
- R2 - Error encapsulation
- R3 - Fault detection
- R4 - Fault identification
- R5 - Code upgrade
- R6 - Stable storage

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

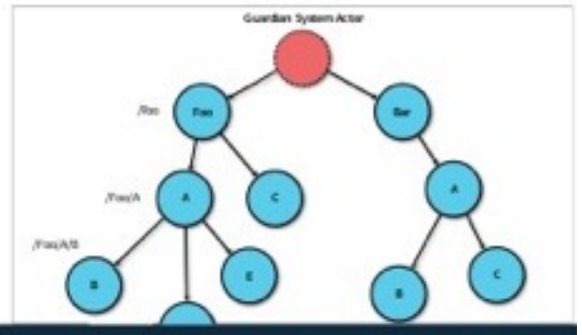



Note: nodes
can be on different
machine

From: Erlang Programming
Cesarini & Thompson 2009

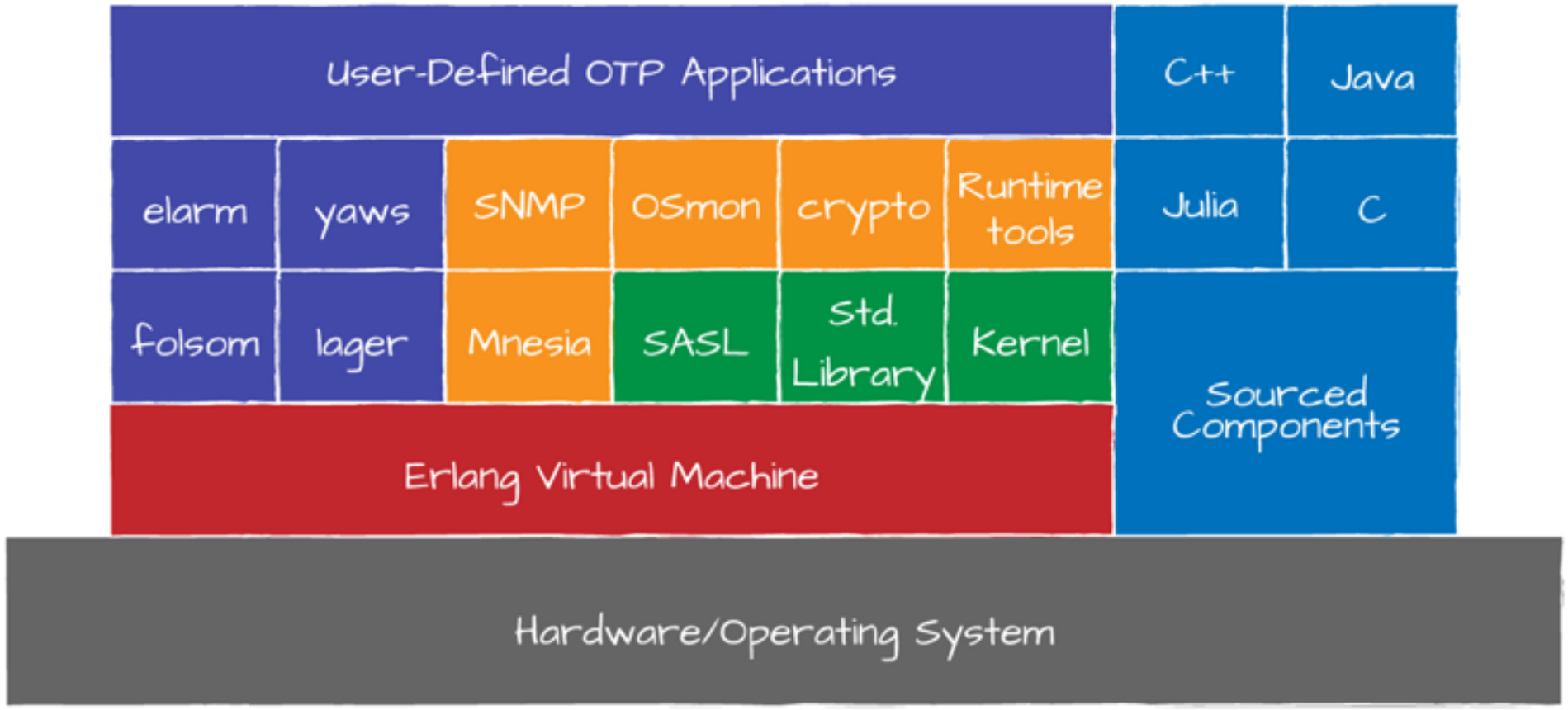
## akka in a few words:

- Toolkit for building scalable distributed / concurrent apps.
- High Performance Actor Model implementation
  - "share nothing" – messaging instead of sharing state
  - millions of msgs, per actor, per second
- Supervision trees – built-in and mandatory
- **Clustering** and **Http** built-in




Typesafe

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

# 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?

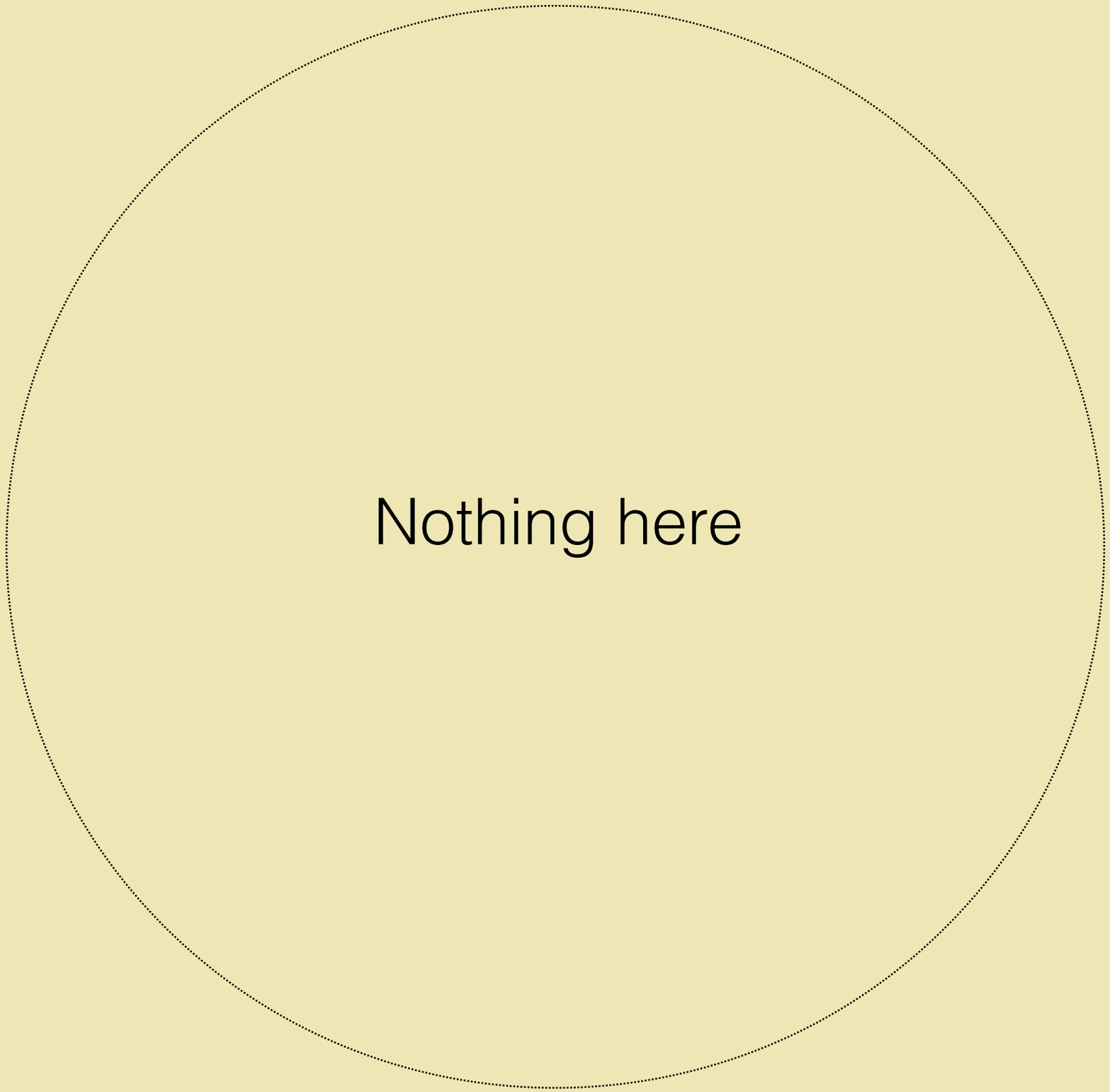
- Ignore it (no)
- Log it (yes)
- Try to fix it (possibly, but don't make matters worse)
- Crash immediately (yes)

In sequential languages with single threads crashing is not widely practised

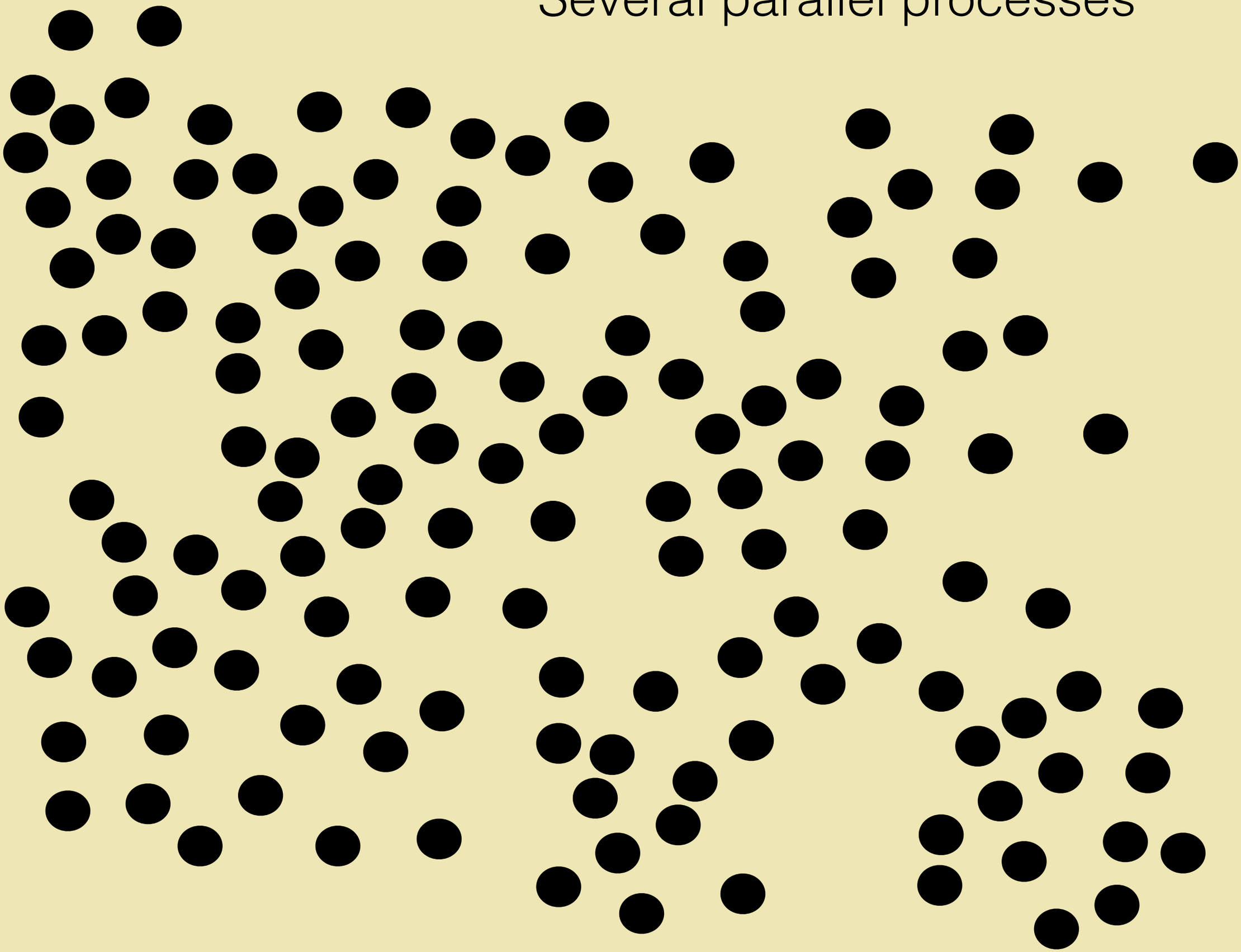
# What's the big deal about concurrency?

# A sequential program

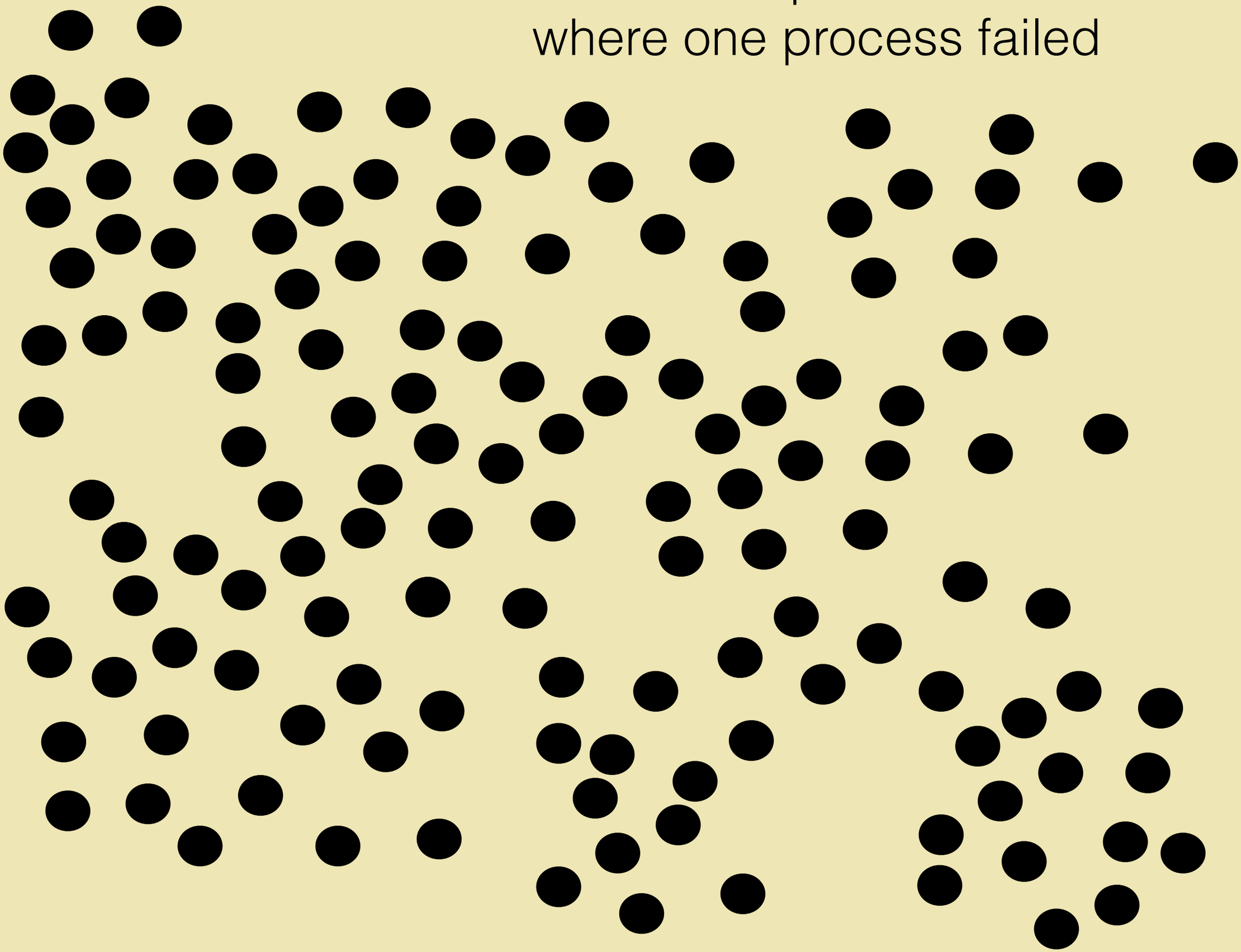
# A dead sequential program



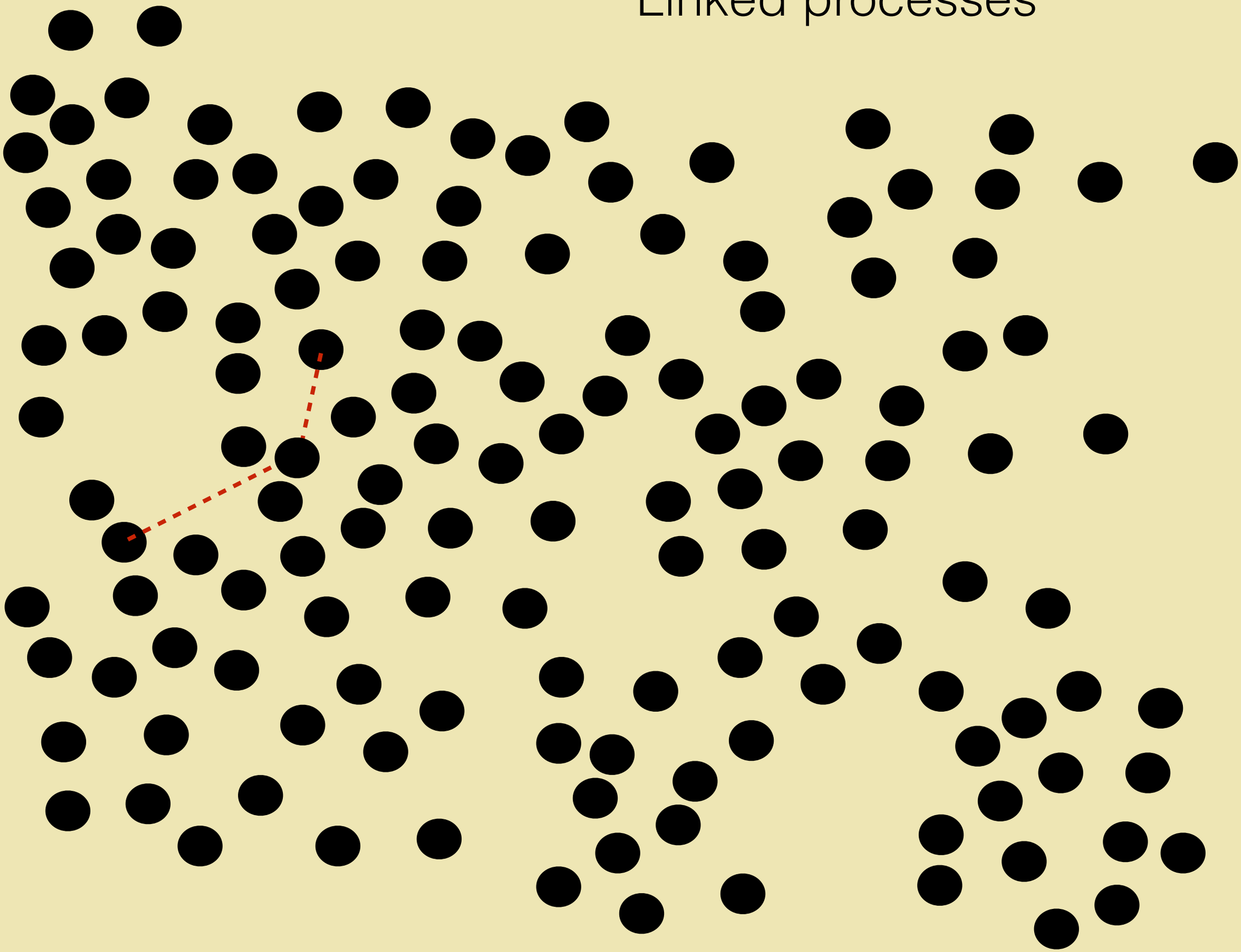
Several parallel processes



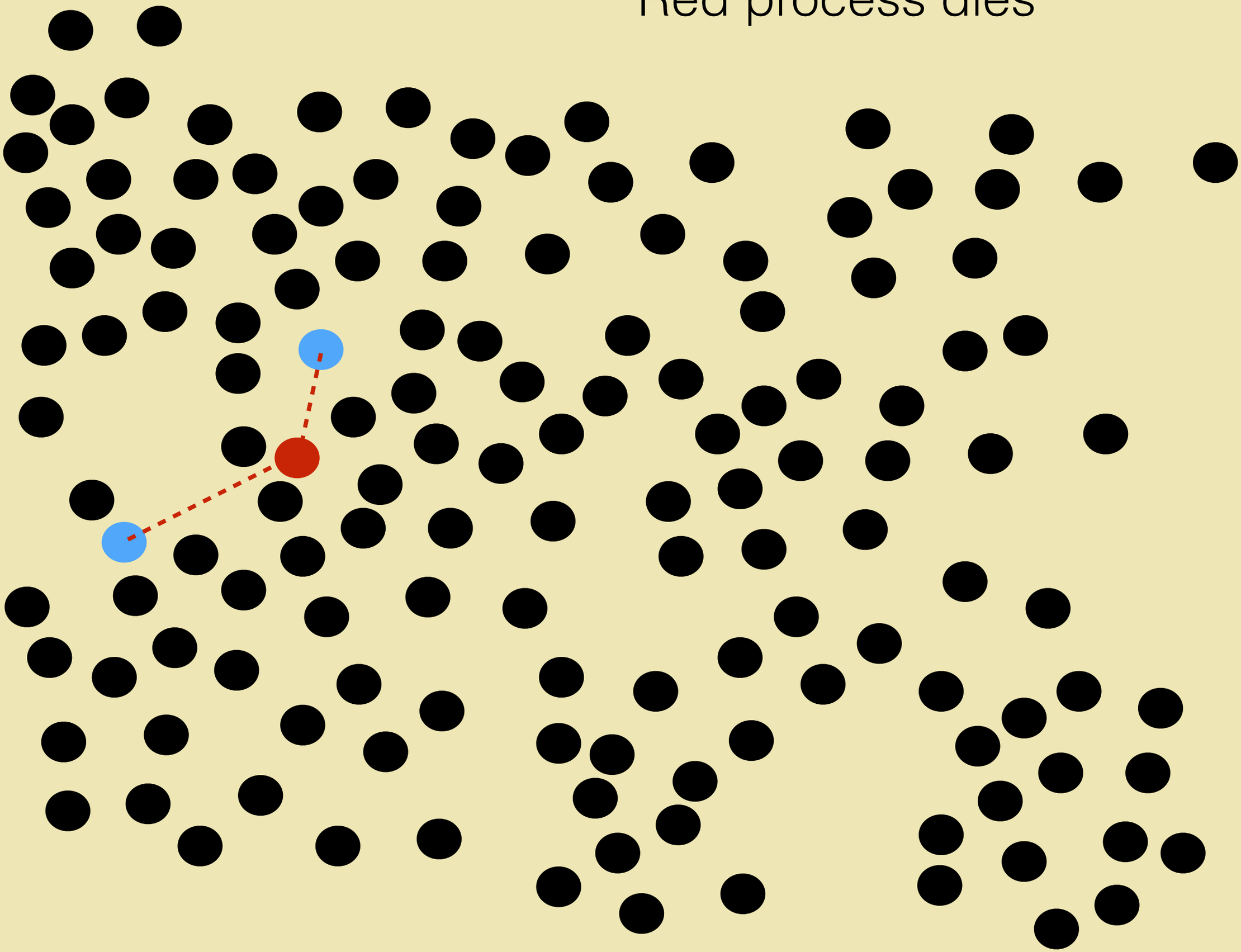
Several processes
where one process failed



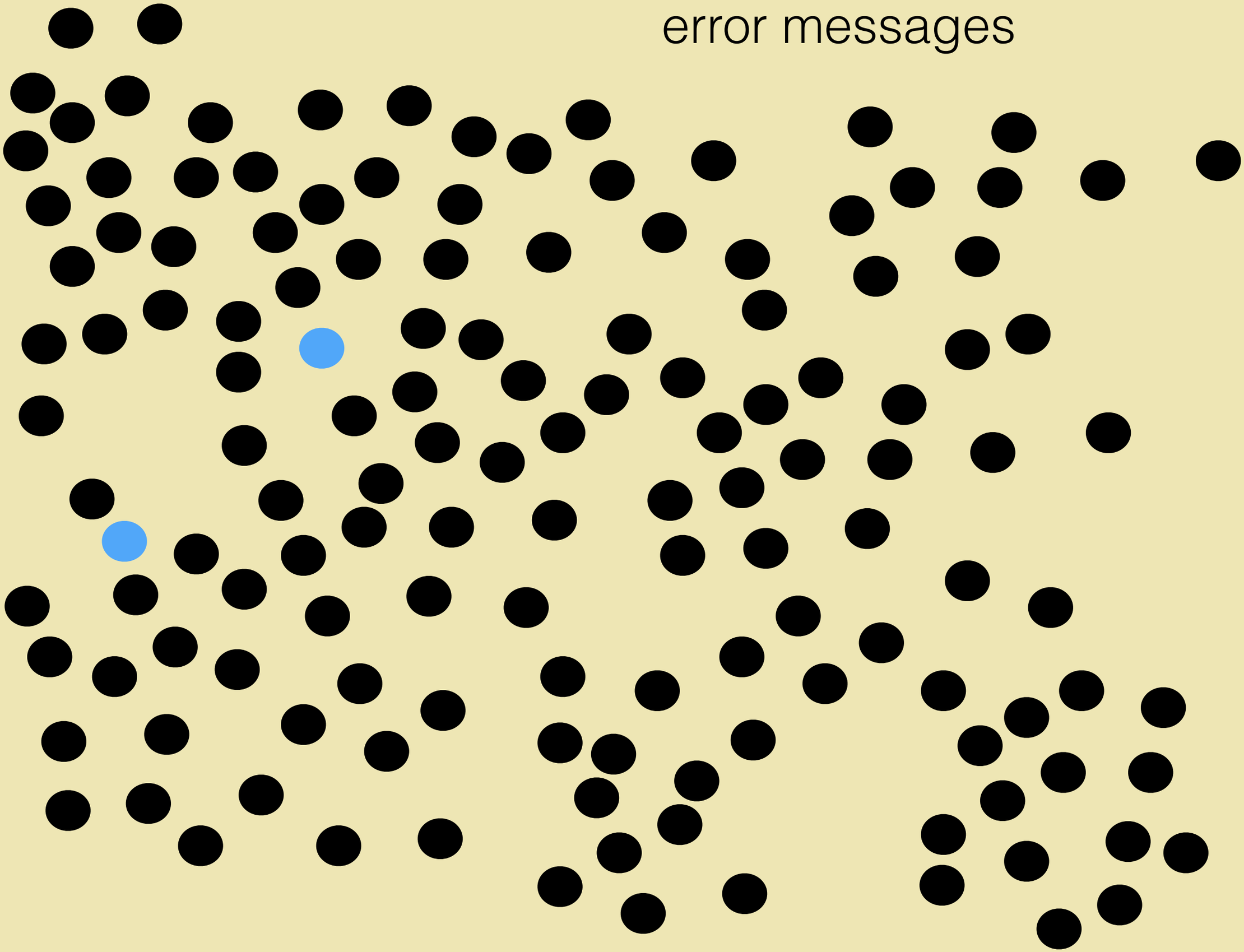
# Linked processes



Red process dies



Blue processes are sent
error messages



AND

Fault-tolerance
is impossible
with one computer

AND

# Scalable is impossible with one computer *

* To more than the capacity of the 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* - *e*rrors 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 aar
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_16S) is
      --\
```

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: $1.18059 \times 10^{21}$
- **Correct answer: –0.827396059946821 36···!**
  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*

0000420002107603560O

# EXPEDITED PARCEL
# COLIS ACCÉLÉRÉS

**2**

**CANADA POST / POSTES CANADA**

*From / Exp.:*

*SretAdd.getFirstName().toUpperCase()*
*SretAdd.getAddressLine1().toUpperCase()*
*SretAdd.getCity().toUpperCase() SretAdd.getState().toUpperCase() SretAdd.g*
*SretAdd.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?

Inside red arrows you find protocols

There are a lot’s of protocols

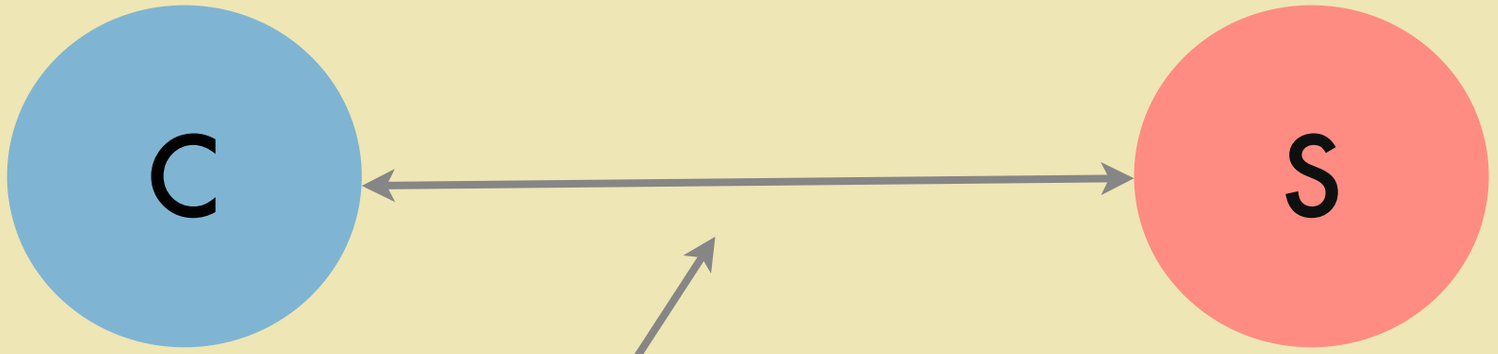
We are incredibly bad at describing protocols

# Protocols are contracts

# Contracts assign blame

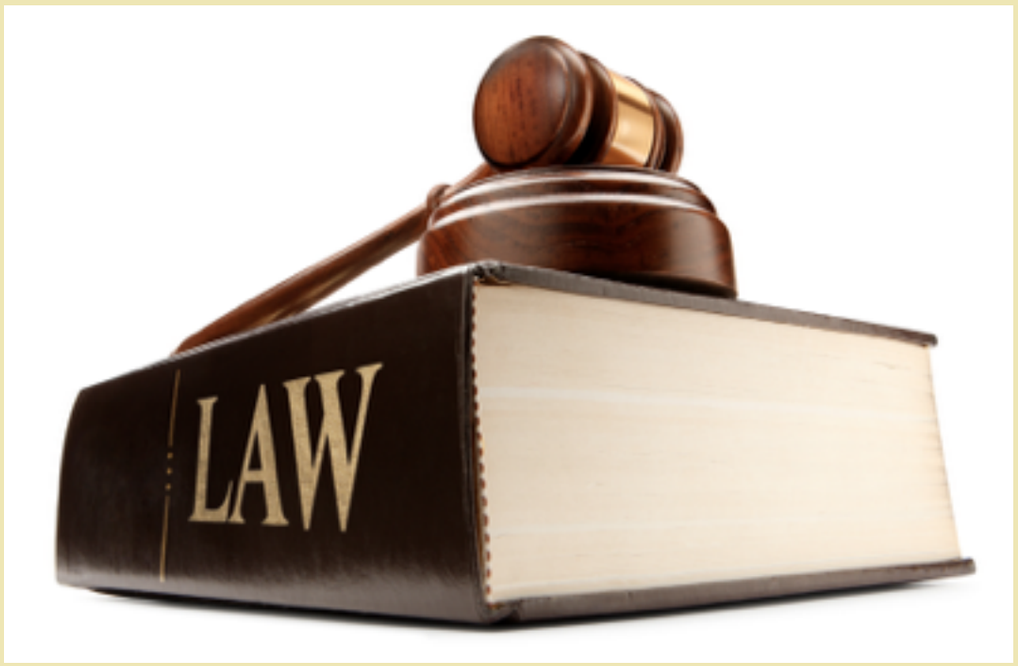

The client and server are isolated by a socket - so it should *"in principle"* be easy to change either the client or server, without changing the other side

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
(First Party)
(Second Party),
WITNESSETH: That in consideration
kept and performed on the
1. Said party of
ovenants and agreements to be
reto, respectively as herein stated:
nts and agrees that it shall:
d said party of the second


The contract checker describes what is seen on the wire.


C
CC
S

# How do we describe contracts?