Multiprocessing — difficult
— doesn't solve the problem

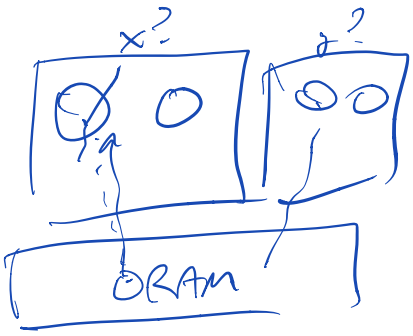
Demand scaling done!

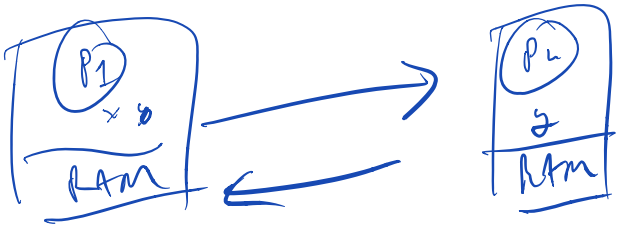
Parallel programming

shared memory (threads)

distributed memory

(processes)





x = 12
z = y

explicit communication

message-passing

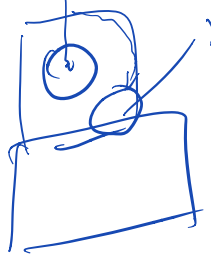
"duals" of each other

machine(p2).sendMessage(p2, "get", "y")

(Lauer & Needham, 1979)

threads processes

```
x=12;
t= new Thread( );
System.out.println(x);
t.wait();
```



x=13; - local variables ("stack")
unshared
- thread local value

C++14
pthread_create
pthread_join

race condition
nondeterminism

deterministic function: $f(x) = f(x)$

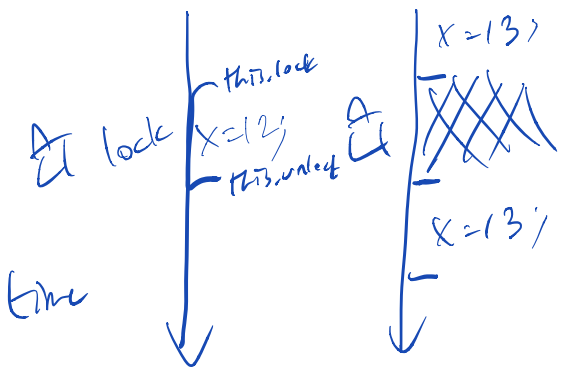
non-deterministic function: $f() \Rightarrow \{12, 13\}$

Control ordering

- wait for threads
- synchronization op
  - prevents concurrency
  - orders events

```
synchronized void foo()
=> this.lock();
   foo()
   this.unlock();
```

lock = mutual exclusion



"mutex"

Condition variable

this.lock()
this.unlock()
} synchronized

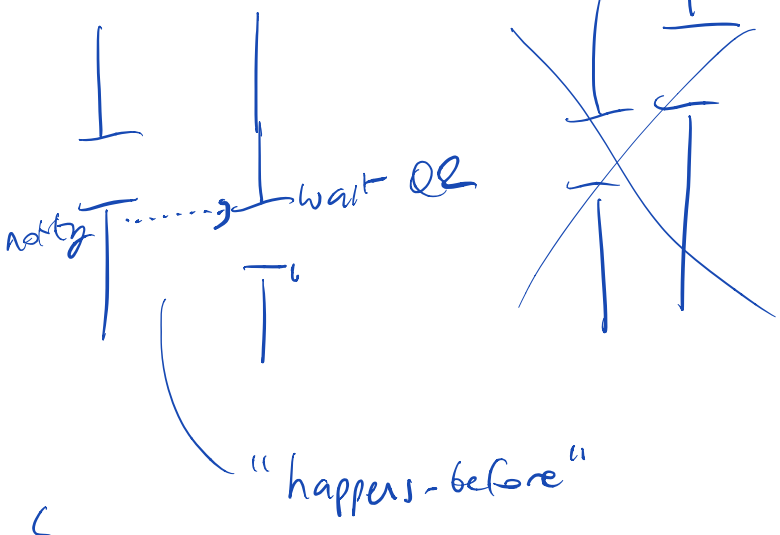
C++

this->lock();
this->lock();

deadlock

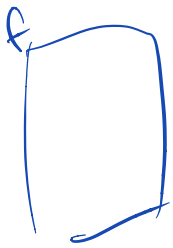
Edsger Dijkstra

wait
notify




```
class Foo {
  public int x;
  synch. void f1()  x=12;
  synch. void f2()  x=13;
}
```

```
synchronized (this) {
  x=12;
}
```



thread 1 thread 2

f.f1(); f.f2();

x = 12; happens-before

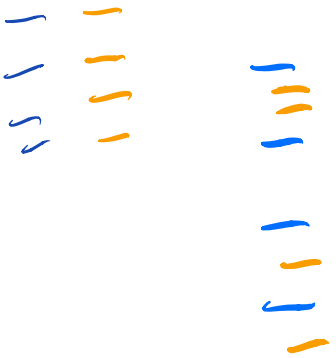
z = 100 end

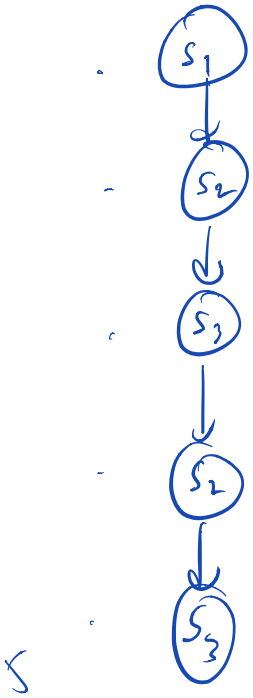
new E1

new E2

Multithreaded challenges:

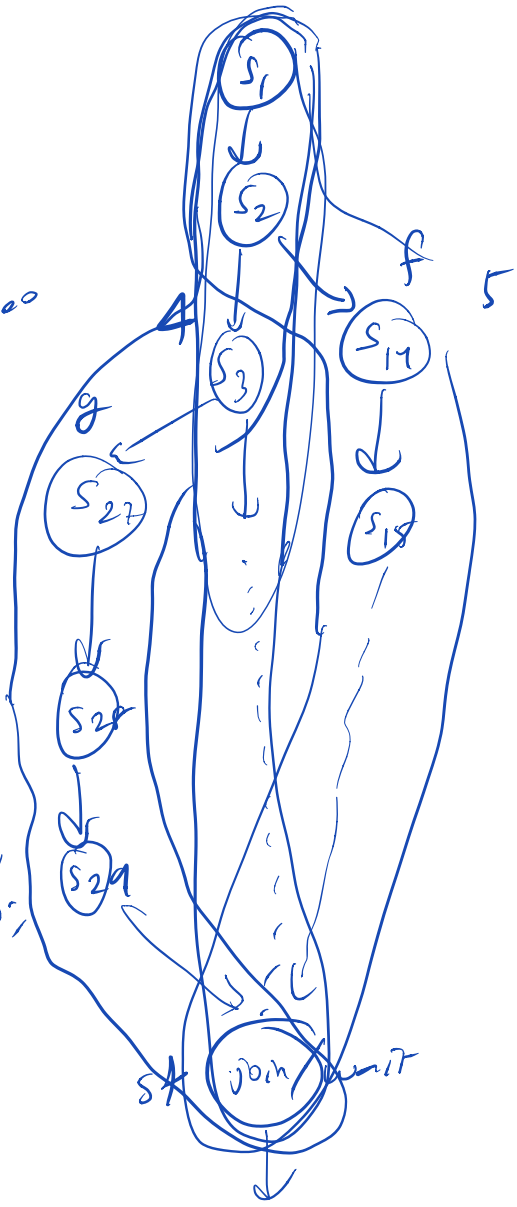
1. correctness — Heisenbugs
2. performance — non-determinism, # potential interleavings





Sequential

```
S1: main() {
S2:   x = 12;
S3:   y = 13;
S4:   z = 1000000
S5:
```




```
S1: main
S2: new Thread f;
S3: new Thread g;
S4: waitForAll();
```

$T_1 = 9$ units "statements"

$\max\{3, 4, 7\} = 7$

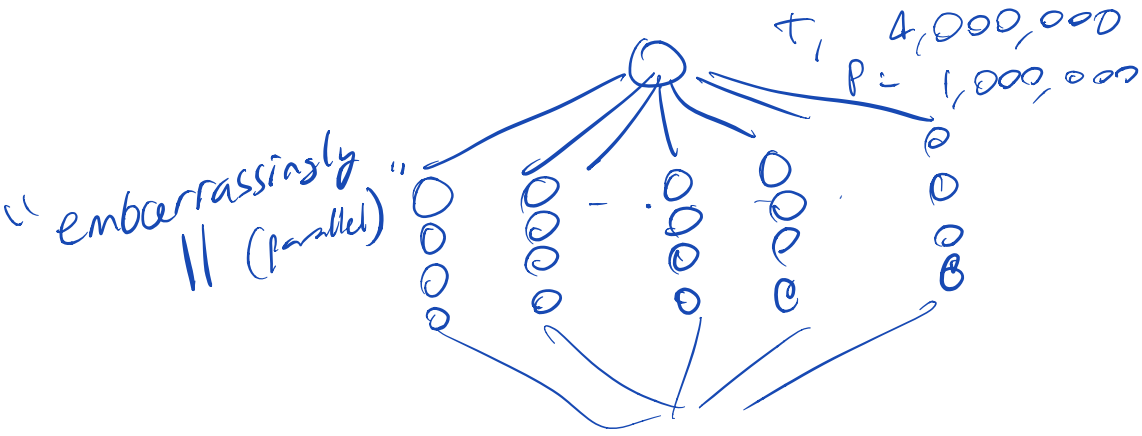
critical path

longest seq. of steps from start to finish

$T_\infty = 7$

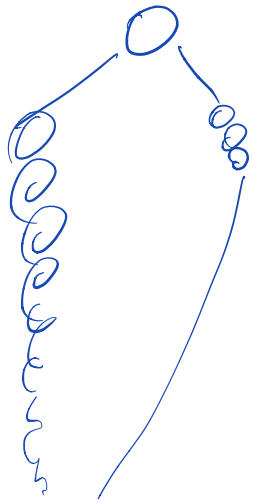
critical path "span"

$$T_P \leq \frac{T_1}{P} + T_\infty$$

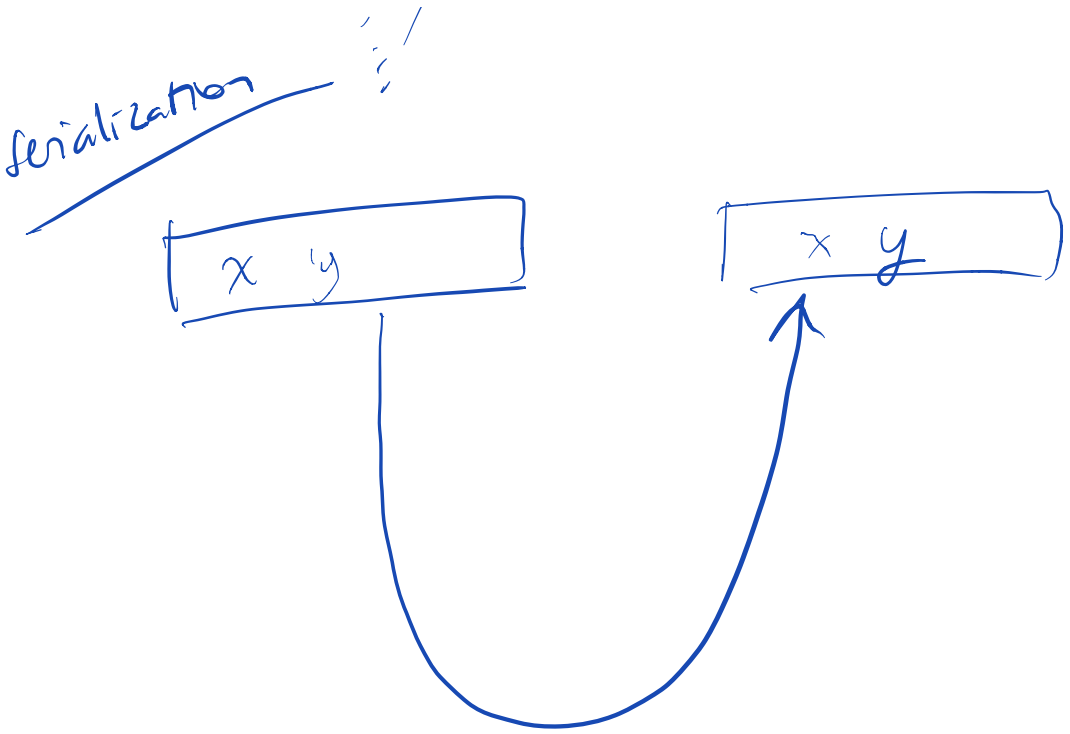


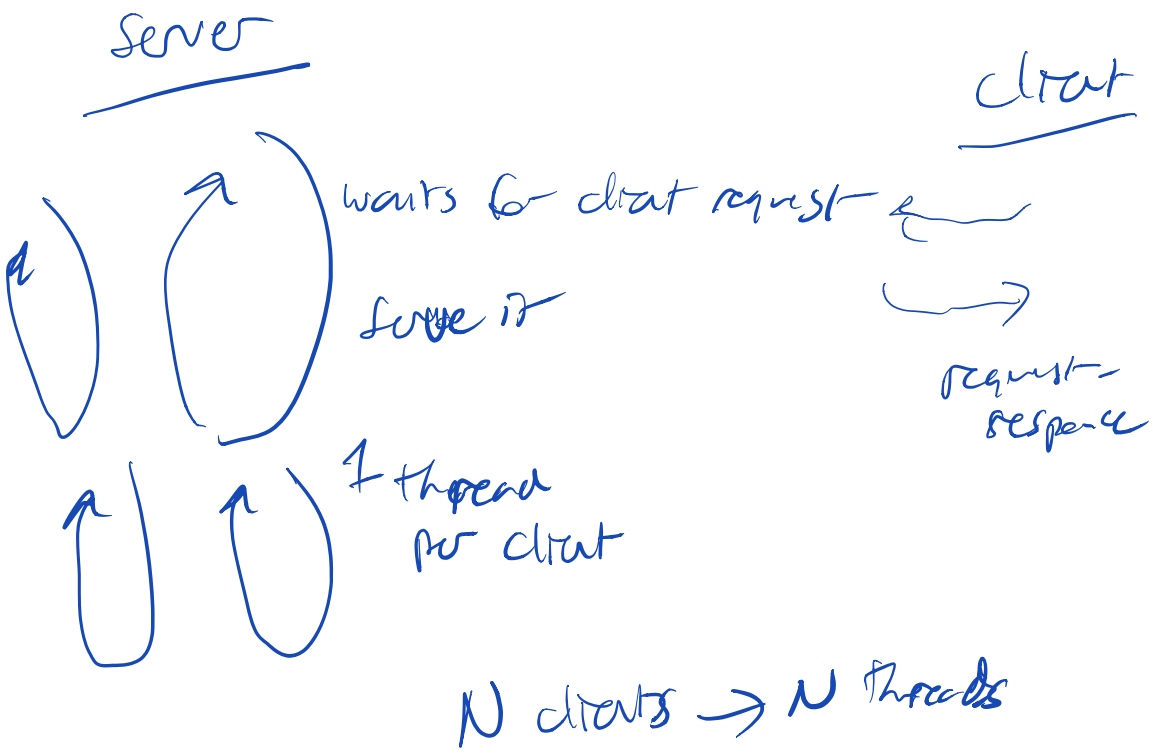
$$\frac{4{,}000{,}000}{1{,}000{,}000} + 4$$

$4 + 4 = P$

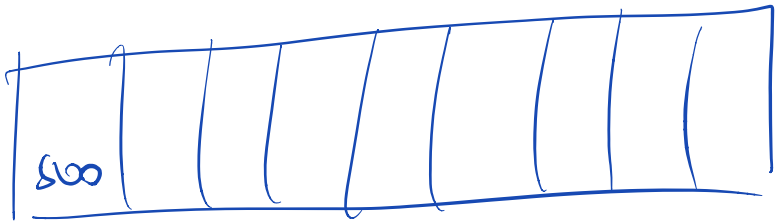


load imbalance


serialization
x y
x y


server
client
waits for client request
serve it
request-response
1 thread per client
N clients → N threads

data



extract word from html

inverted index

amazon: 500, 1435, 999
android: 500, 217
prime: 500, 144

"Gustafson's Law" in action

"big data"
>> 1TB

Thread 1	Thread 2	Share?
local variables (stack variables)		✗
heap?		✓
file handles		✓
globals?		✓
code		✓

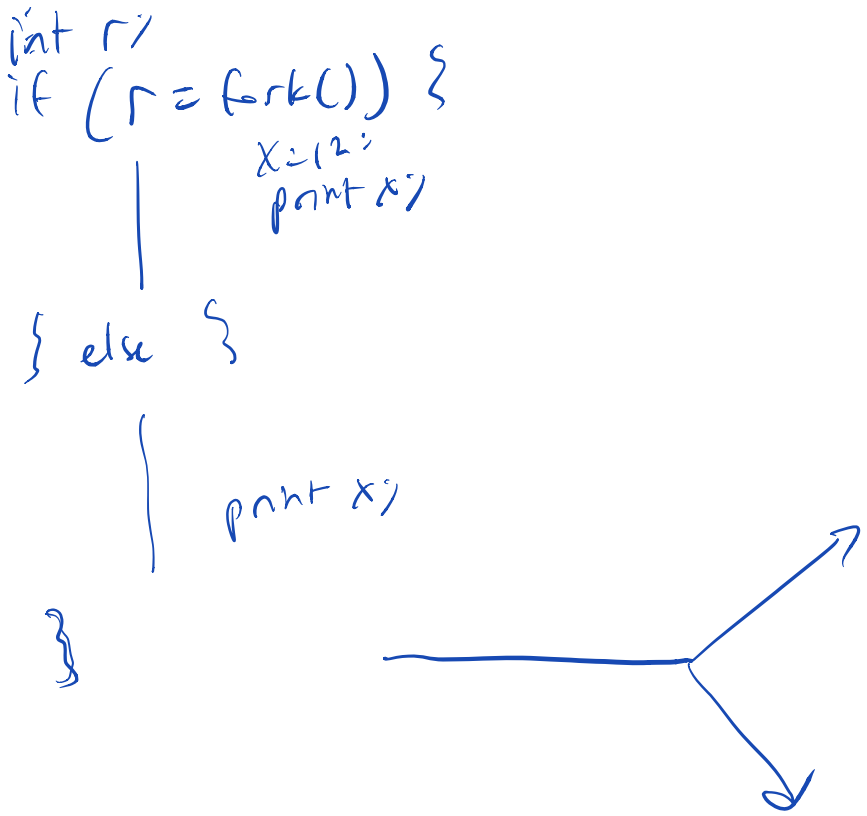
process 1	process 2	Share?
local variables (stack variables)		✗
heap?		✗
file handles		✗
globals?		✗
code		✓

spawn(p)

fork()

clone()

int x = 0;


int r;
if (r = fork()) {
x=12;
print x;
} else {
print x;
}