# 并发编程

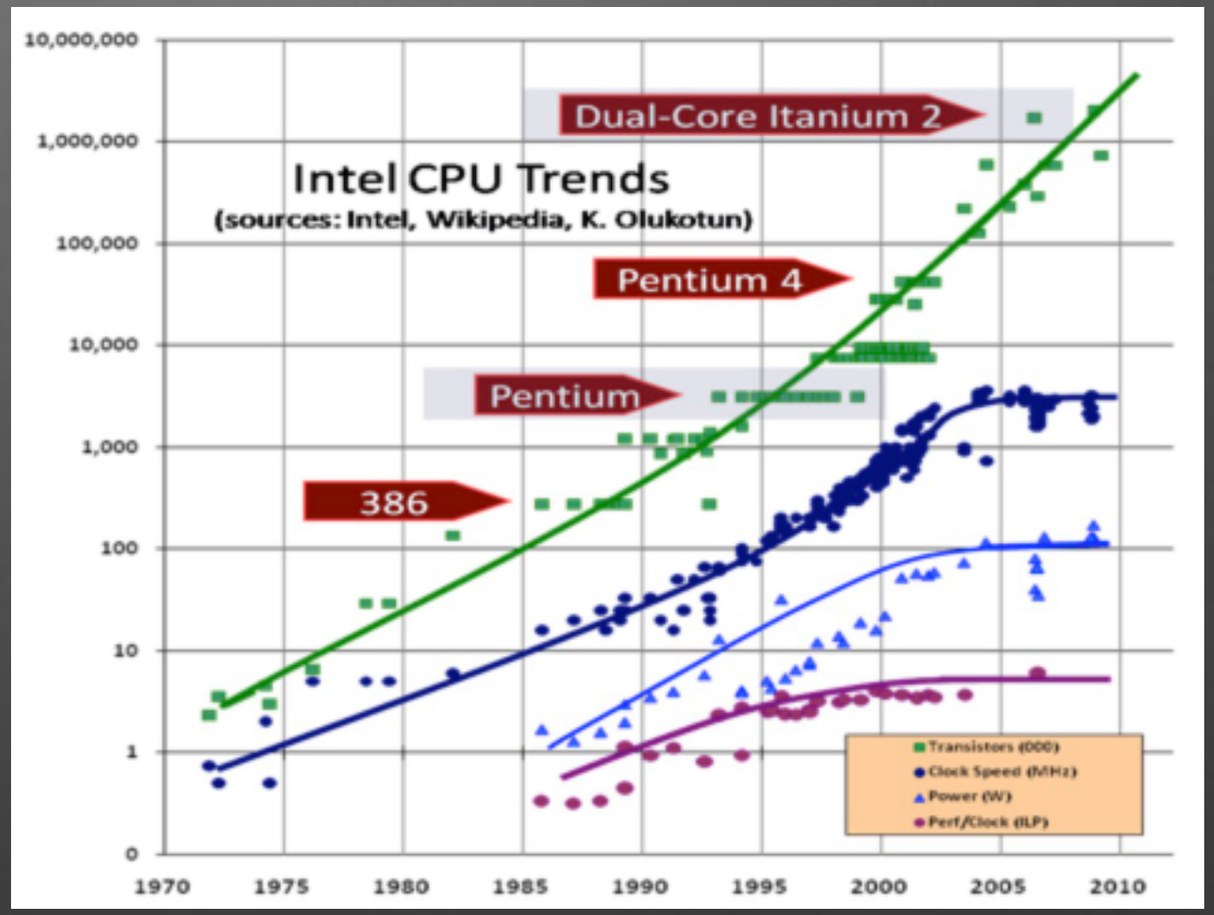
sunkai

# 趋势

摩尔定律

多核危机

Amdahl定律




Erlang Haskell Clojure Go Scala

# 并发 vs 并行

并发是同一时间应对（**dealing with**）多件事情的能力；

并行是同一时间动手做（**doing**）多件事情的能力；

—— **Rod Pike**

# 并行 不等于 多核

Bit-Level

Instruction-Level

Data-Level

Task-Level

# 并发 不只是 多核&多线程

性能

高效

容错

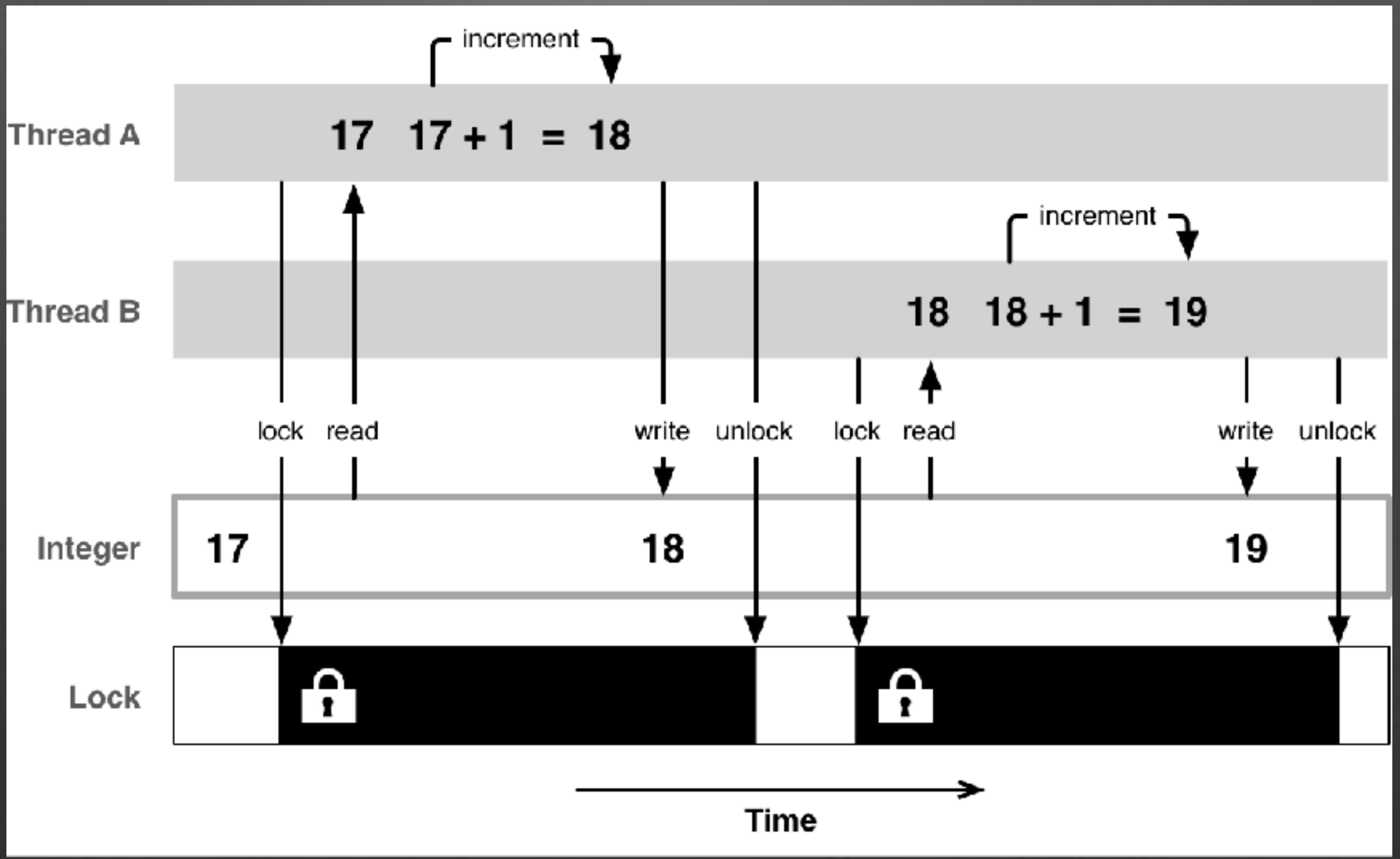
简单

# 聊聊并发模型

# 线程 & 锁

# 线程 & 锁

- 底层硬件运行过程的形式化
- 并发的基本单元—线程
- 线程间通过**共享内存**来通信


increment
Thread A
17
17 + 1 = 18
increment
Thread B
18
18 + 1 = 19
lock
read
write
unlock
lock
read
write
unlock
Integer
17
18
19
Lock
Time

# 线程 & 锁

- 竞争条件
- 乱序
- 可见性

# 线程 & 锁

- 语言级解决方案
  - 内存模型 ie. Java Memory Model (JSR 133)
  - happens-before
  - as-if-serial

- 其他问题
  - 死锁
  - 活锁

# 复杂性

- 可中断锁
- 可超时锁
- 交替锁
- 分段锁
- Condition

- ThreadPool
- CopyOnWrite
- BlockingQueue
- Compare and swap

可变状态 + 并发 = 不确定性

可变性是灾难！

# 函数式编程

# 函数式编程

- 命令式编程 vs 函数式编程
  - 计算过程，转换成改变全局状态的指令
  - 计算过程，抽象为表达式（纯函数）求值

# 函数式编程

- y = f(x)
  - input: x
  - output: y
- 无副作用的纯函数，无可变状态，天然支持并发
- x=x+1　FAIL

# 函数式编程

- Immutable data
- First class functions
- Tail recursion
- Currying
- ...

# 函数式编程

```
list.parallelStream
    .map(this::retrieveFromA)
    .map(this::processUsingB)
    .forEach(this::saveToC)
```

Java 8 Streams

# STM

# 函数式编程 - Clojure

- Lisp方言
- base JVM
- 混合函数式编程 & 可变状态

# Clojure - STM

- 软事务内存，支持**ACI**，~~没有D~~
- 基于**MVCC**
  - 所有对ref的读在tx开始时，看到一个一致的snapshot及tx对ref进行的修改
  - tx内对ref的修改，外部看到的是一个时间点的触发
- 写冲突
  - abort & retry
  - barge
- 无死锁&活锁

# Clojure - STM

- 四种模型
  - ref
  - atomic
  - agent
  - vars

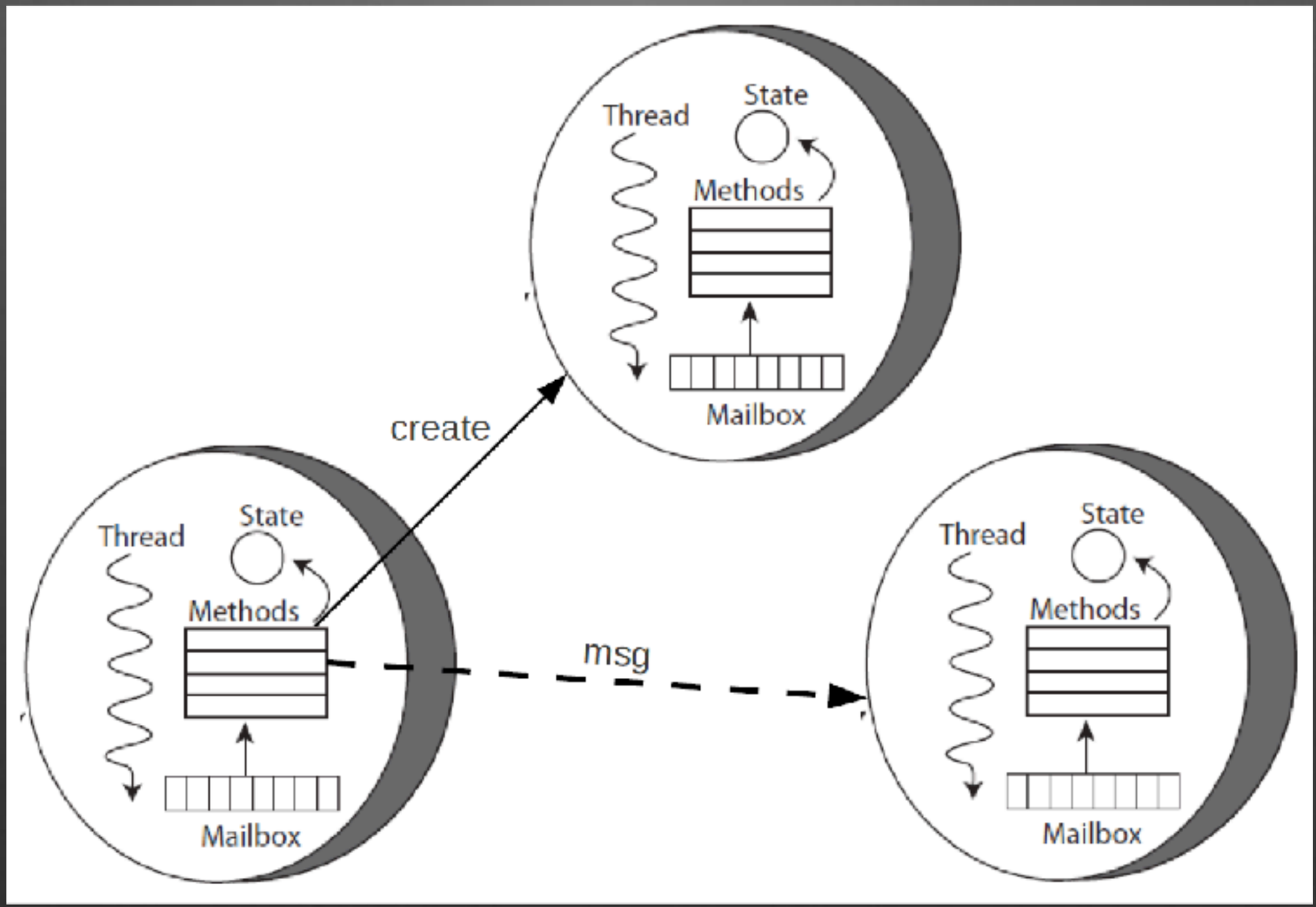
| name | Coordinated/Independent | Synchronous/Asynchronous |
|---|---|---|
| Ref | Coordinated | Sync |
| Atomic | Independent | Sync |
| Agent | Independent | Async |
| Vars | ThreadLocal | Sync |

# Actor

# Actor

- Hewitt 1977
- 万物都是Actor
  - Actor之间只能发送消息
  - **Actor=数据+行为+消息**
- Coroutine/Green Thread

# Actor


Thread
State
Methods
Mailbox
create
Thread
State
Methods
Mailbox
msg
Thread
State
Methods
Mailbox

# Akka

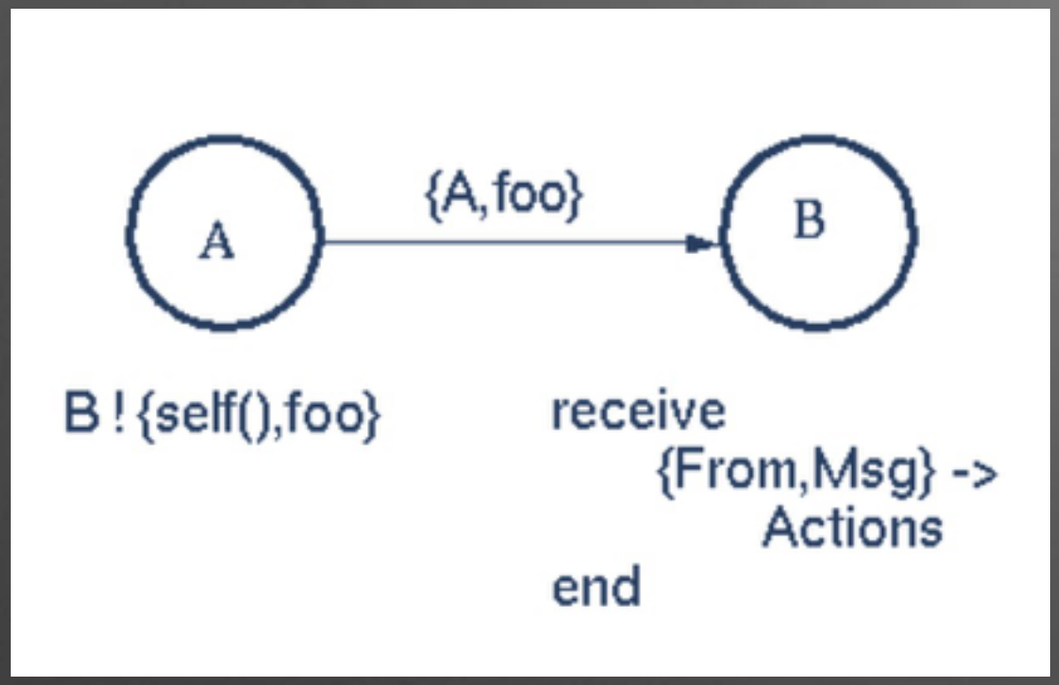
- 基于Scala
- 支持Scala & Java
- Let it crash
- Never stop

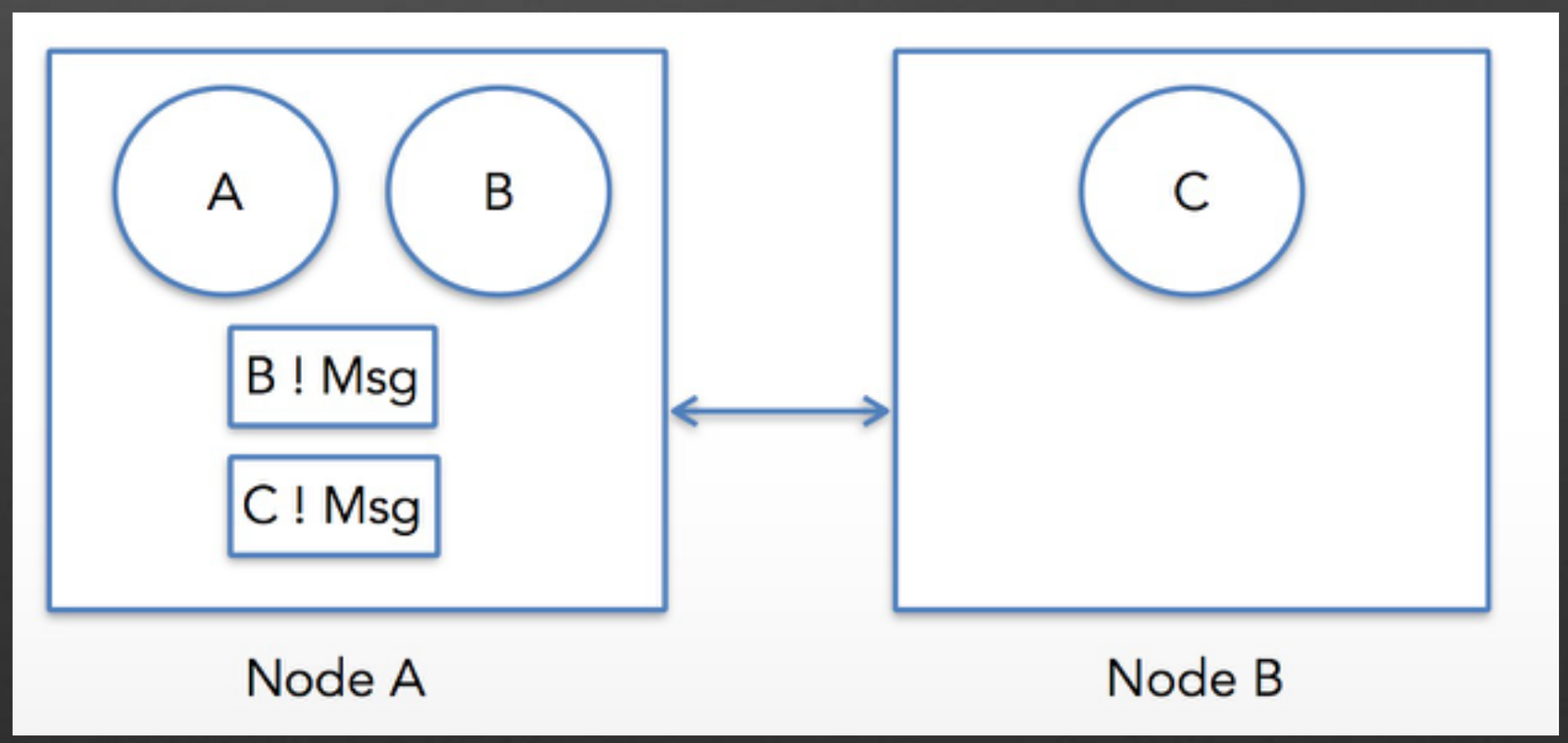
http://developer.lightbend.com/guides/akka-quickstart-java/full-example.html

# Erlang

- 函数式语言
- 轻量级进程　独立身份，独立状态， Actor
- 强占式调度　Reduction
- 进程独立GC
- 分布式

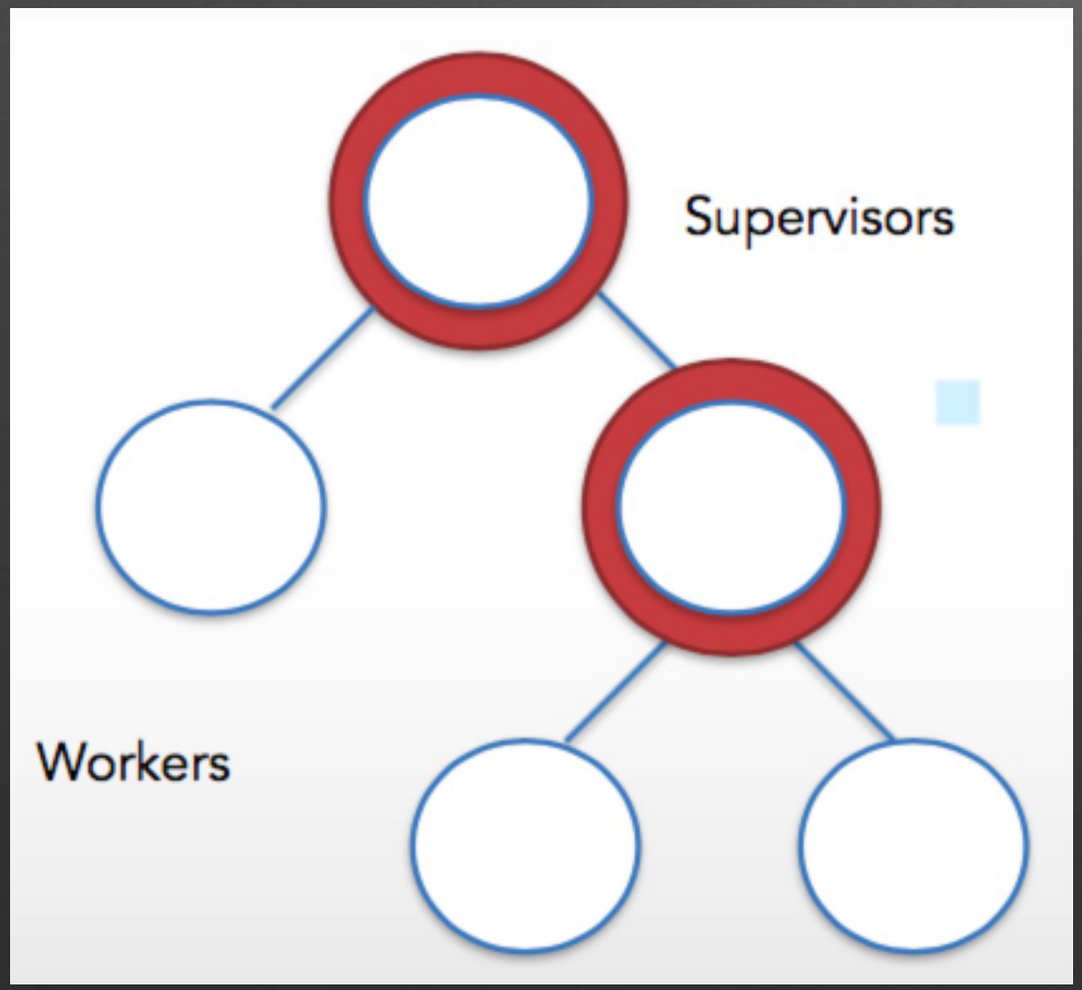
# Erlang

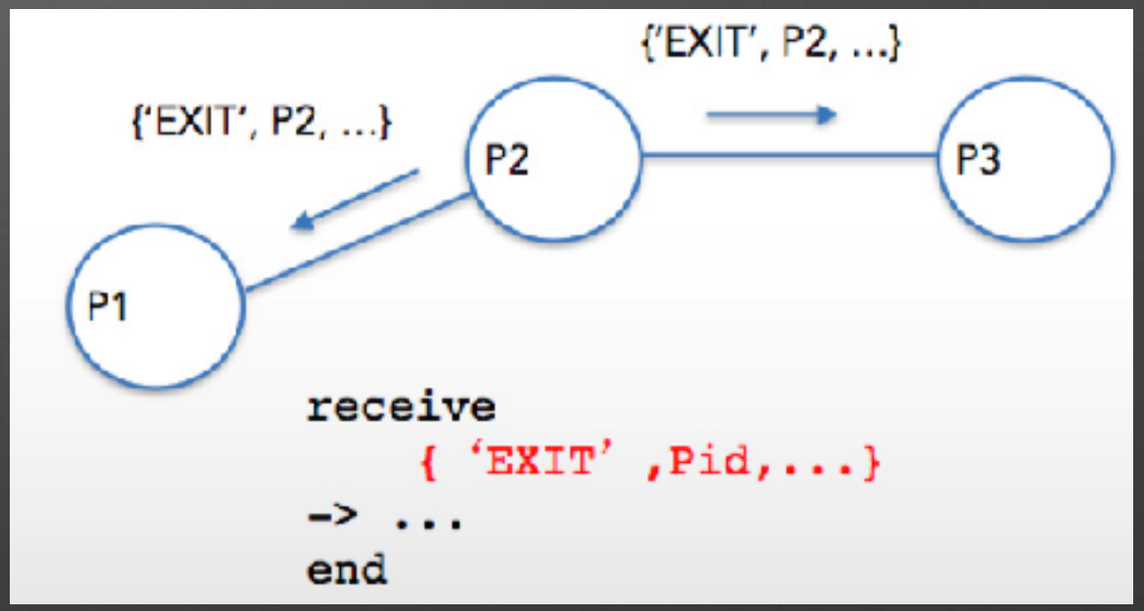

A
{A,foo}
B
B ! {self(),foo}
receive
{From,Msg} ->
Actions
end


A
B
B ! Msg
C ! Msg
C
Node A
Node B

# Erlang

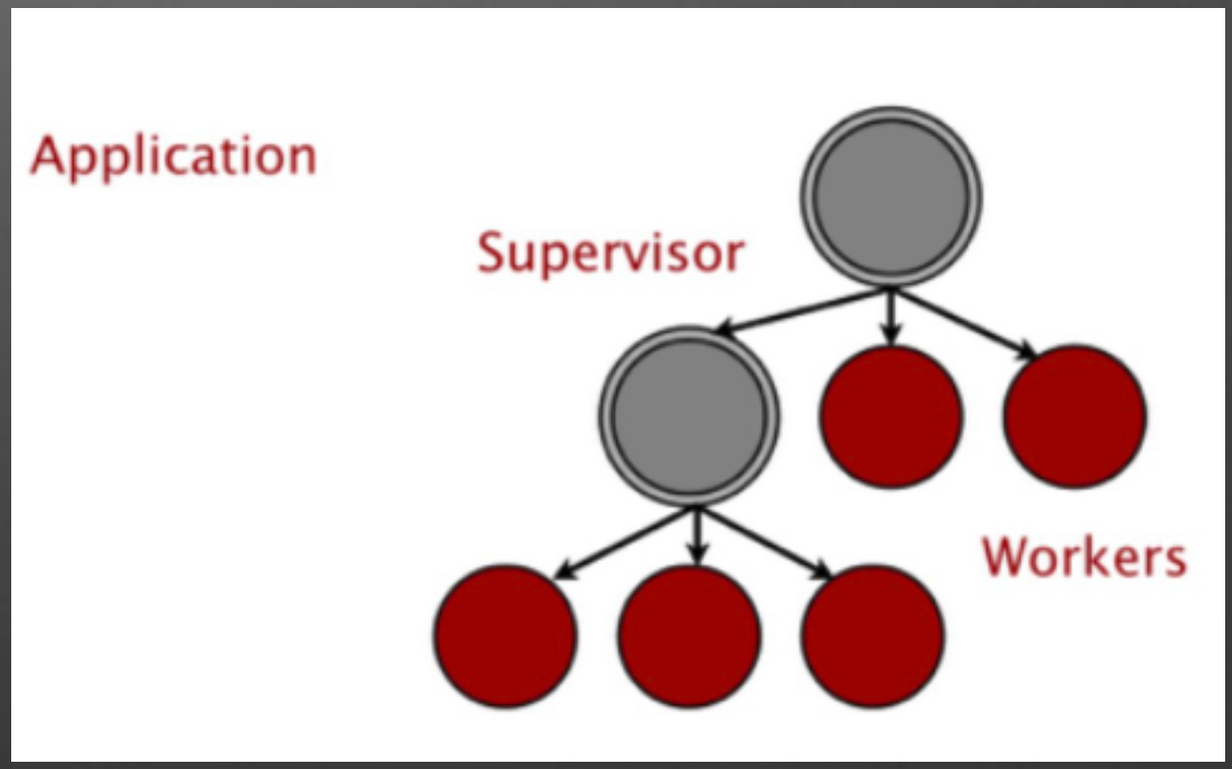
- Let it crash
- Fail-Fast



# Erlang OTP

- Application
- Supervisor
- Worker
  - gen_server
  - gen_event
  - gen_fsm

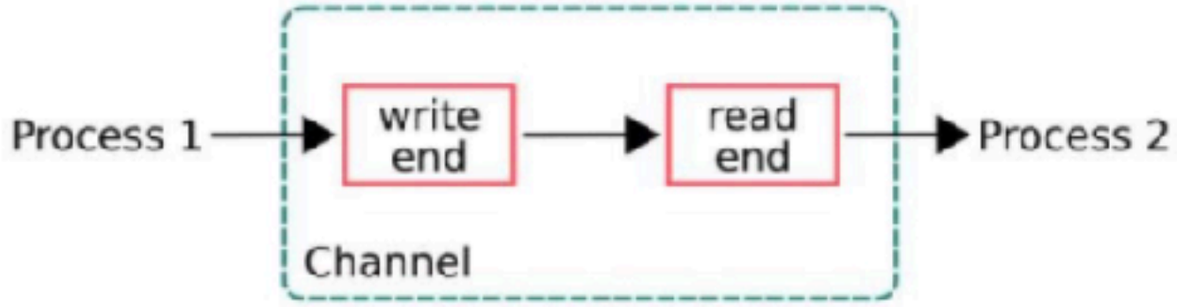


# CSP

# CSP

- R. Hoare 1978
- Communication Sequential Processing

# CSP vs Actor




CSP

- Communication through channels
- Processes are "anonymous"


Actor Model

- Point-to-point communication
- No anonymity

# 其他

- Future Promise

- Callback

# 推荐

- 七周七并发
- Java并发编程实战
- Java虚拟机并发编程
- http://www.erlang-cn.com/
- http://akka.io/