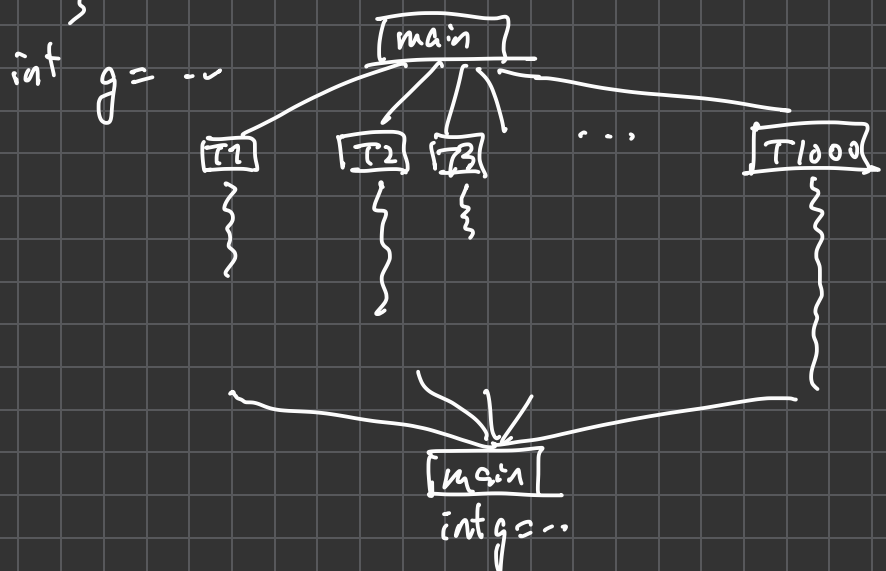
Concurrency, Asynchronous Programming

1. At a single time step, multiple controls of computation are going simultaneously

---

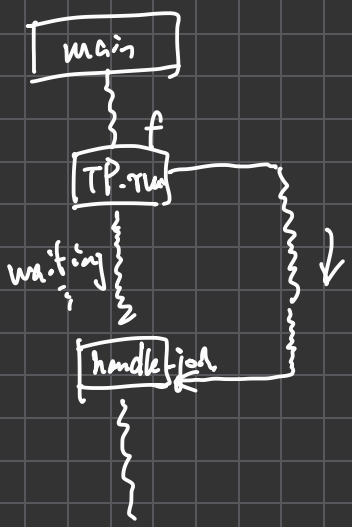
C/C++

```
#pragma amp for
for (int i=0; i<1000; i++) {
    // ... body
}
int g= ...
```

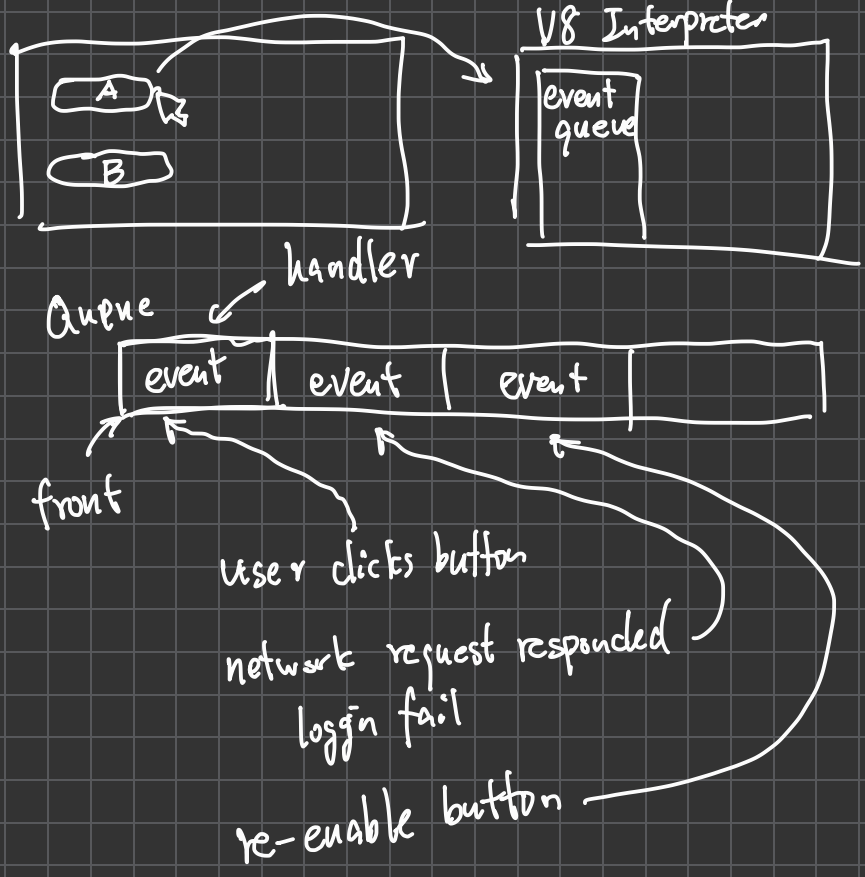


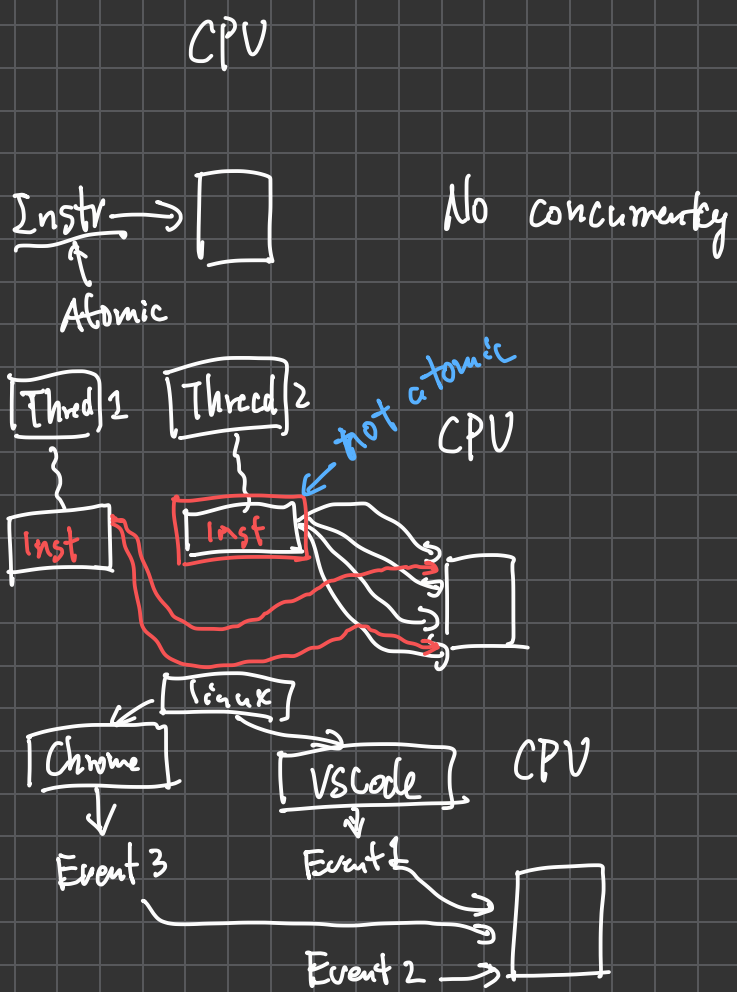
Python

```
def f() : ...
pool = ThreadPool(...)
handle = pool.run(f)
handle.join()
```



JS.

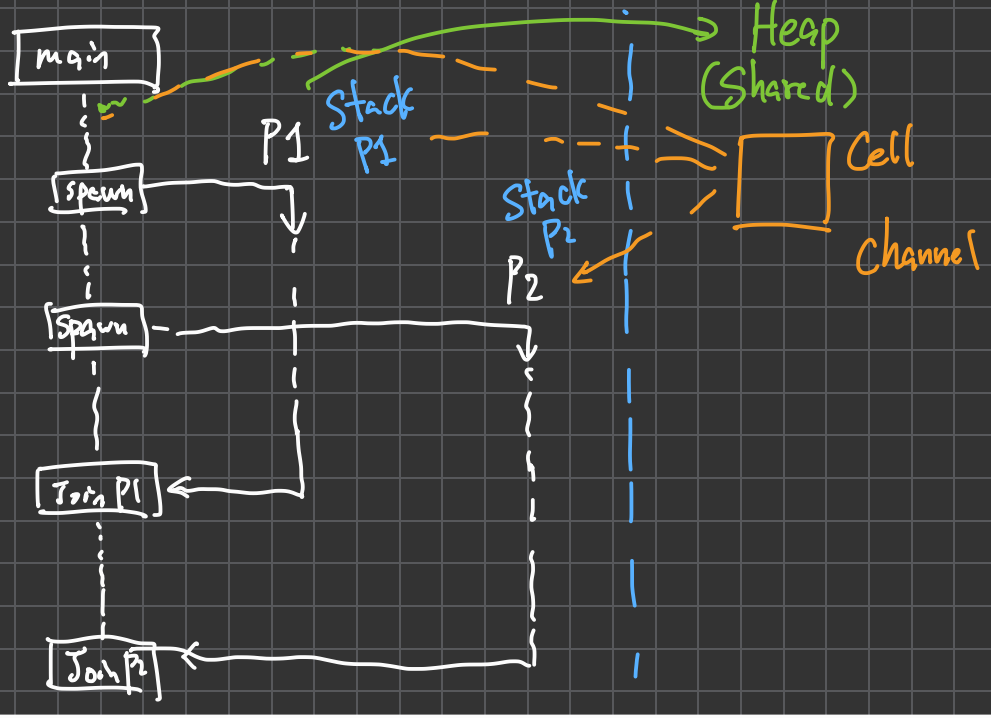




Properties of Concurrency

1. Spawning new controls
2. Communicating between controls

C communication



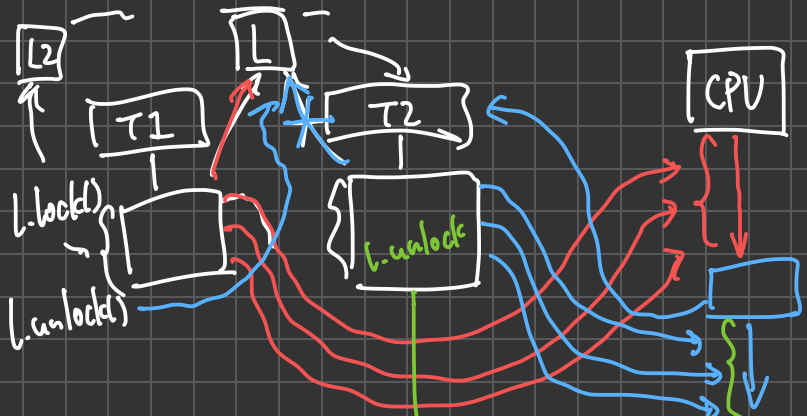
1. Data Racing → Atomic Operation

Atomic Region

Lock

mutual exclusion

mutex_lock



Rust

Ownership

Brrow

2. Deadlock

everyone is waiting

F^b

Big-step semantics

bad for describing concurent programs

$e \Rightarrow v$

input — output value

AF^b

Asynchronous

small-step

$G_0 \to G_1 \to G_2 \to \cdots$

Two-layer frame

Top-layer

Global-layer

$G_i \to G_{i+1}$

1. Can reason about diverging programmes

Idle $G \to G$ reflexivity

---

big step

$e \xRightarrow{\Delta G} v$ ← Side effects

Local layer

Atomic computation

Actor-Model

1. Communicate via Message Passing

2. Spawn other Actors

$A F^b V$

- A: Async
- F^b: Variant

$$
\begin{aligned}
v &::= (F^b\ V\ \text{value} \ldots) \\
&\mid a \leftarrow \text{name of the actor} \\
e &::= (F^b\ V\ \text{expr} \ldots) \\
&\mid \text{Create}(e, e)
\end{aligned}
$$

Create(e, e): Spawn actor, initial msg

behavior / function

$G : \text{Soup}$

$\{\langle a, f \rangle \ldots\}$

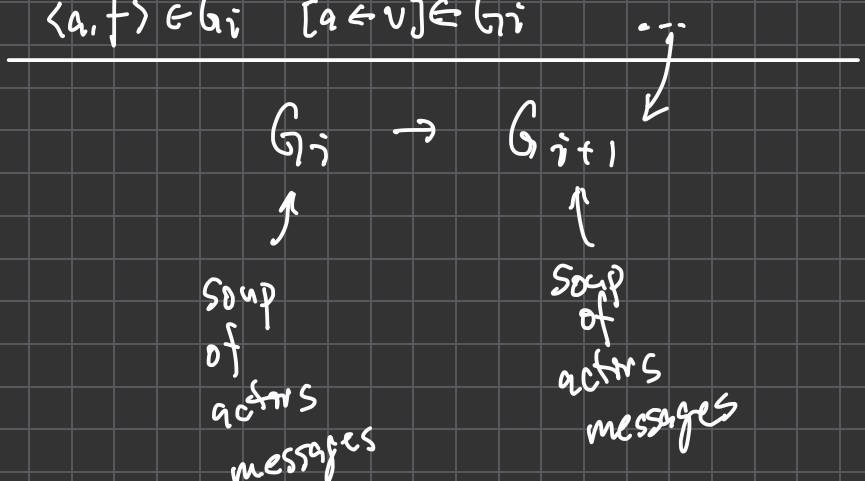
$\langle a, f \rangle$: actor name, behavior

$\mid e \leftarrow e$

$e \leftarrow e$: send msg; actor's name, msg

$U \{[a \leftarrow v] \ldots\}$

$[a \leftarrow v]$: actor, message


⟨a.f⟩ ∈ Gi
[a ← v] ∈ Gi
...
Gi → Gi+1
soup of actors messages
soup of actors messages