

PCAD

Programmazione Concorrente

Algoritmi Distribuiti

Arnaud Sangnier
arnaud.sangnier@unige.it

INTRODUZIONE

Informazioni

- **Due corsi a settimana:**

- Martedì: 11:00 -> 13:00 in aula 505
- Mercoledì: 11:00 -> 13:00 in aula 505

- **Laboratori:**

- A volte il mercoledì: sarete avvisati il martedì prima in lezione e su Aulaweb

- **Pagina aulaweb:**

- <https://2024.aulaweb.unige.it/course/view.php?id=2406>

- **Valutazione:**

- Un esame scritto di 2 ore e punti bonus (massimo 3) se consegnerete i laboratori (potete farli in gruppo di massimo 3 studenti)

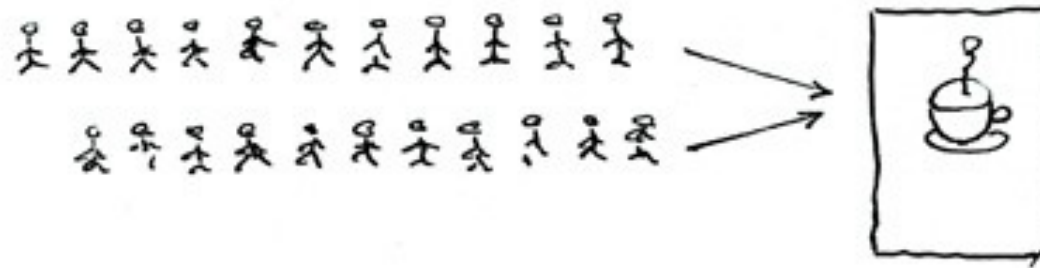
Comunicazione

- Mi potete scrivere a: arnaud.angnier@unige.it
- Leggo le miei mail ogni giorno e rispondo entro 24 ore (lavorative)
 - Rispettare le regole di buon educazione nelle vostre mail
 - Non dimenticate di **firmarle** e di specificare per quale materia mi scrivete etc
- **Non mandatemi programmi scritti nel corpo delle mail !!!! -> Mandate il codice in file separati**

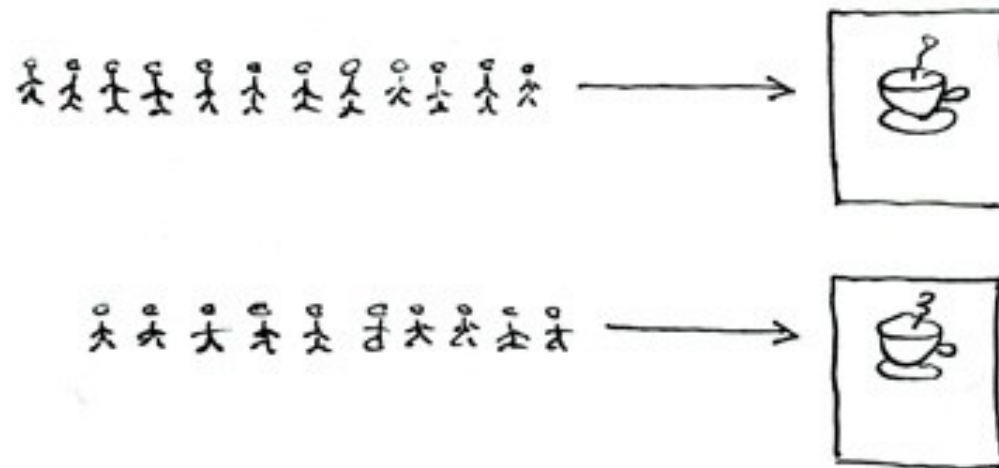
Obiettivi del corso

- Capire che cosa è la concorrenza nei sistemi informatici
- Capire perché è complessa
- Imparare le soluzioni esistenti per scrivere programmi concorrenti
- Imparare i paradigmi della programmazione concorrente
- Imparare algoritmi classici e le loro prove di correttezze
- Scrivere applicazioni distribuite in C e in Java e Golang

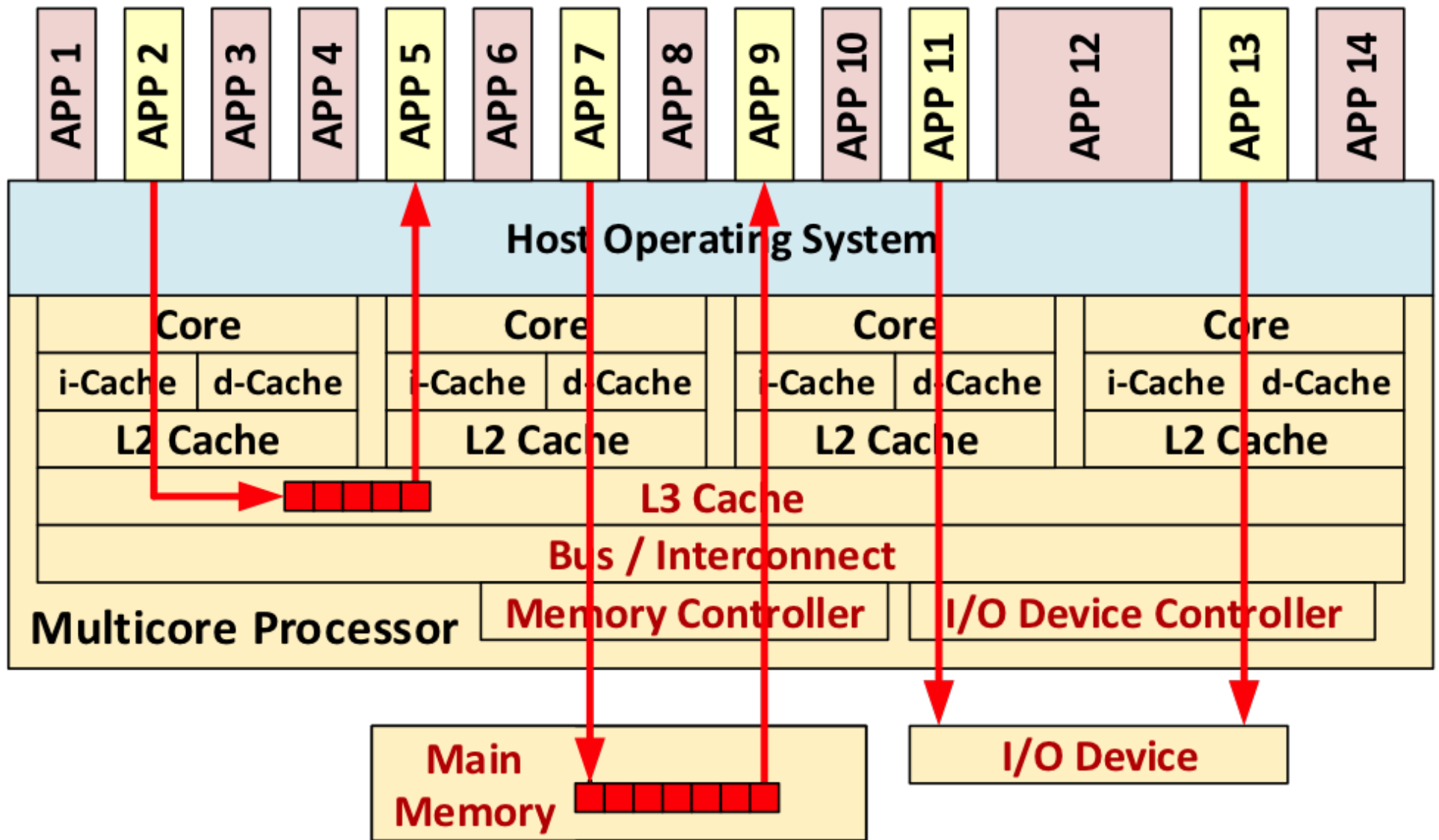
Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines

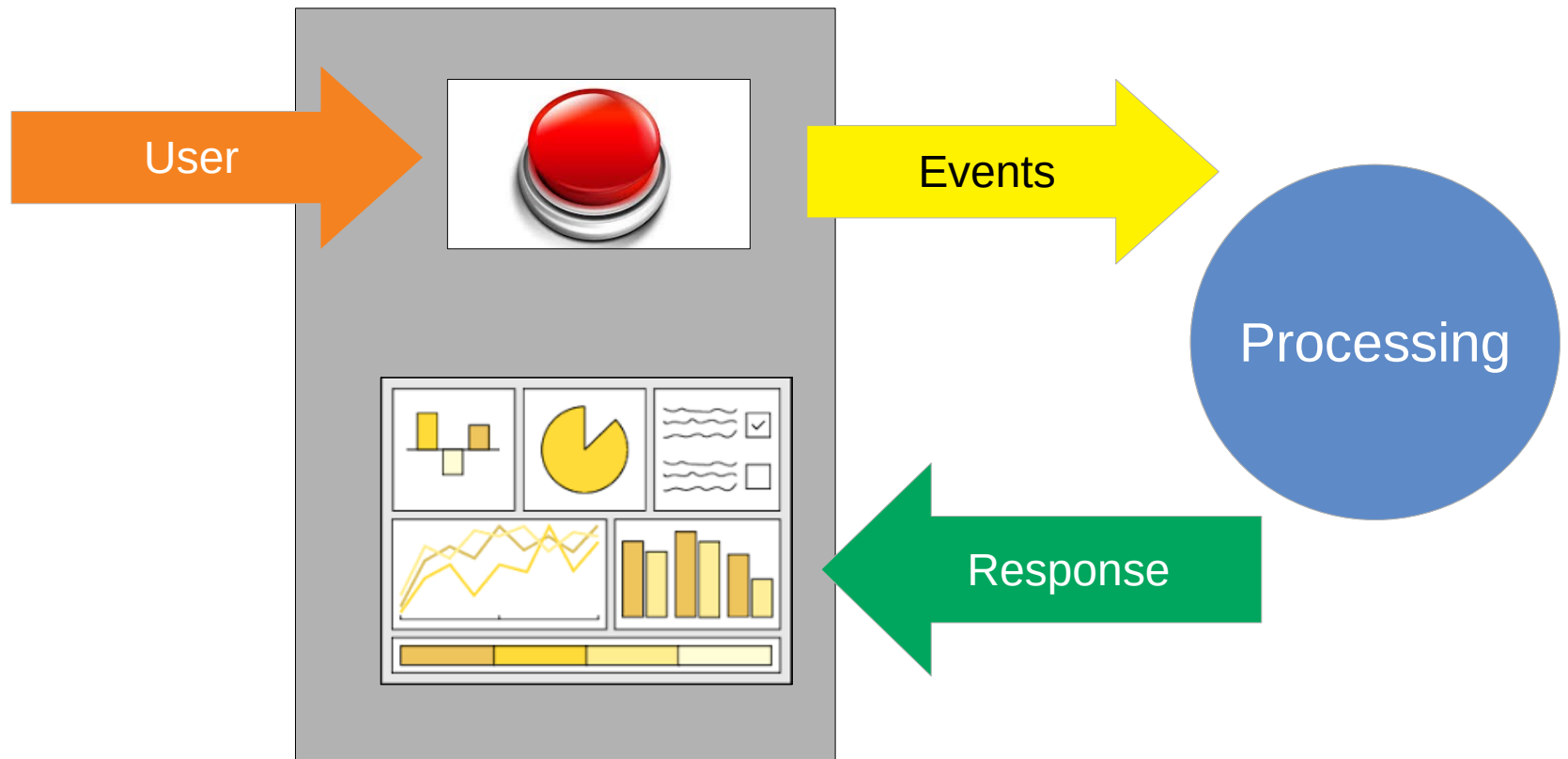


© Joe Armstrong 2013

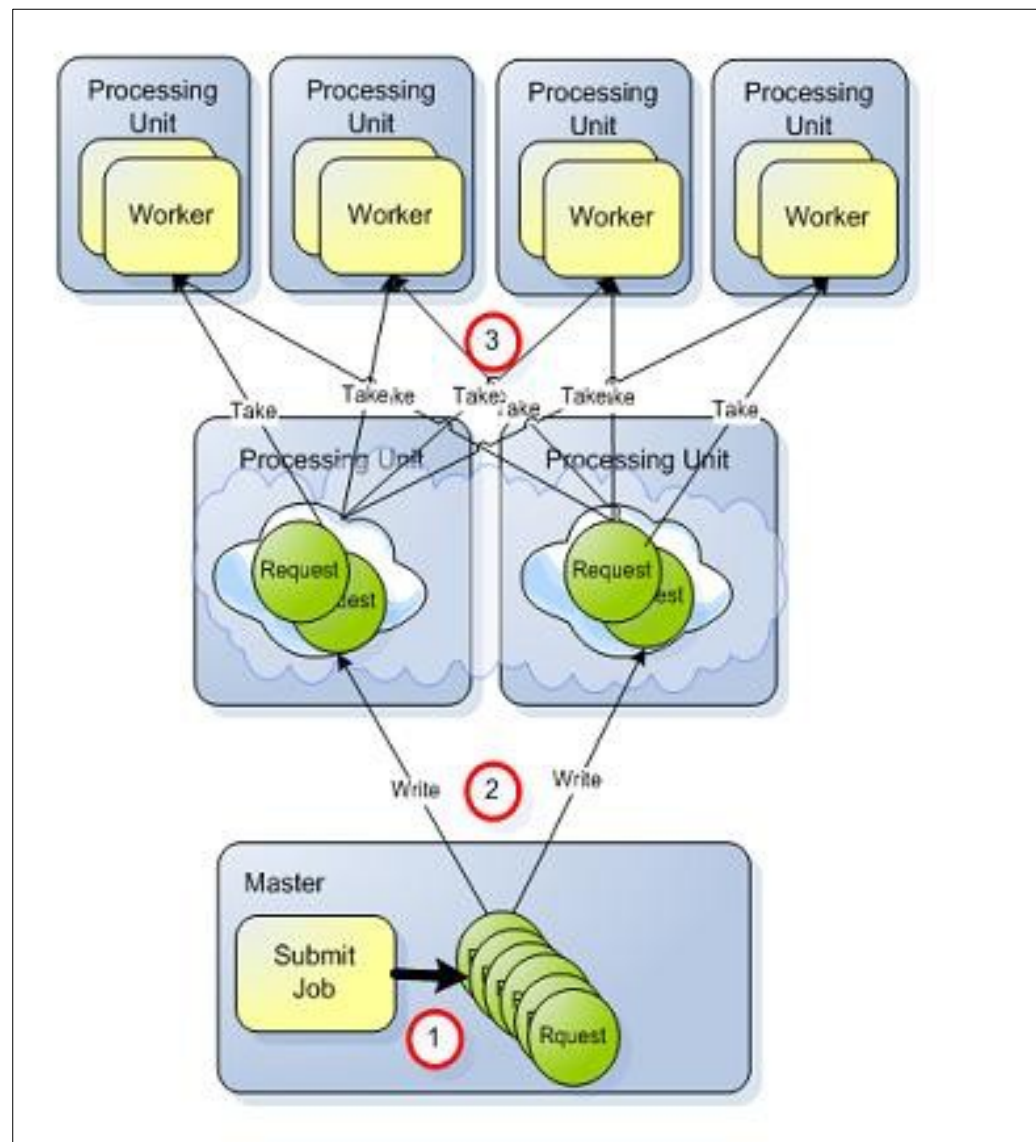


Architettura multicore

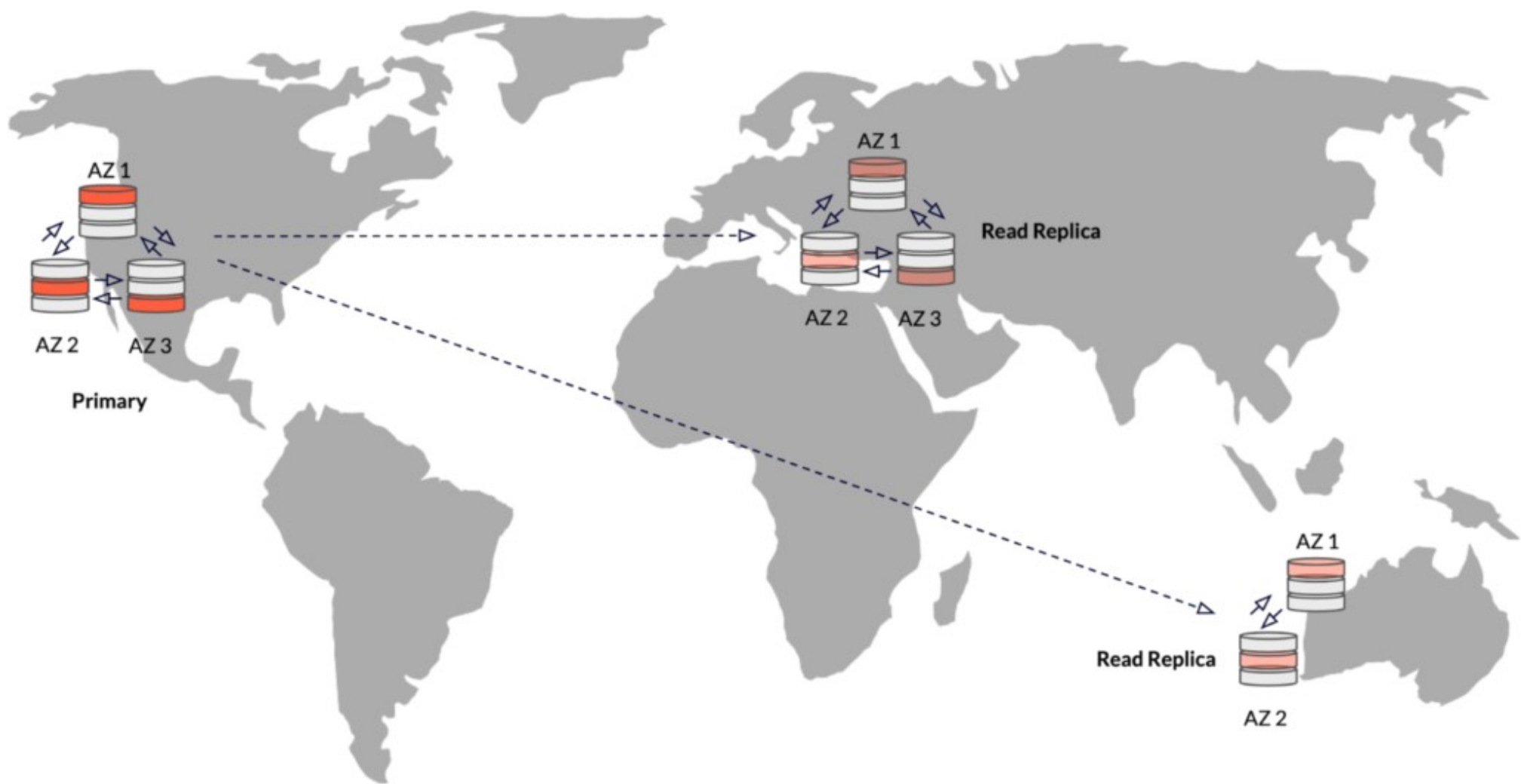
Browser/GUI



Model View Controller (MVC)



Computing Cluster



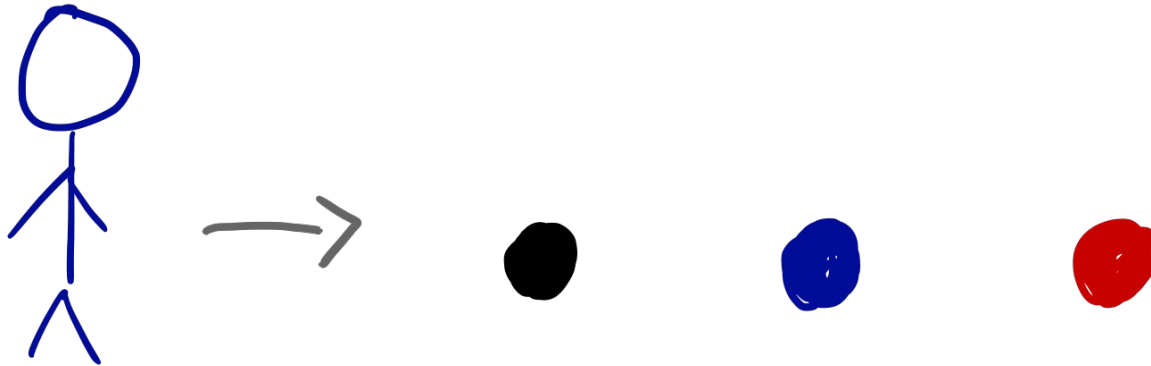
Replicated database

Perché ?

- Perché usare/sviluppare sistemi concorrenti/distribuiti?
 - Per essere più efficiente
 - Per distribuire dei dati, condividere delle risorse, per usare reti di computer
 - Per comunicare
- Fine della Legge di Moore
 - “il numero di transistor nei processori raddoppia ogni 18 mesi”
- Ora non possiamo più aggiungerne per problemi di limite fisici
- Già negli anni 2000, un processore non bastava più ed eravamo passati ad architetture multi processori

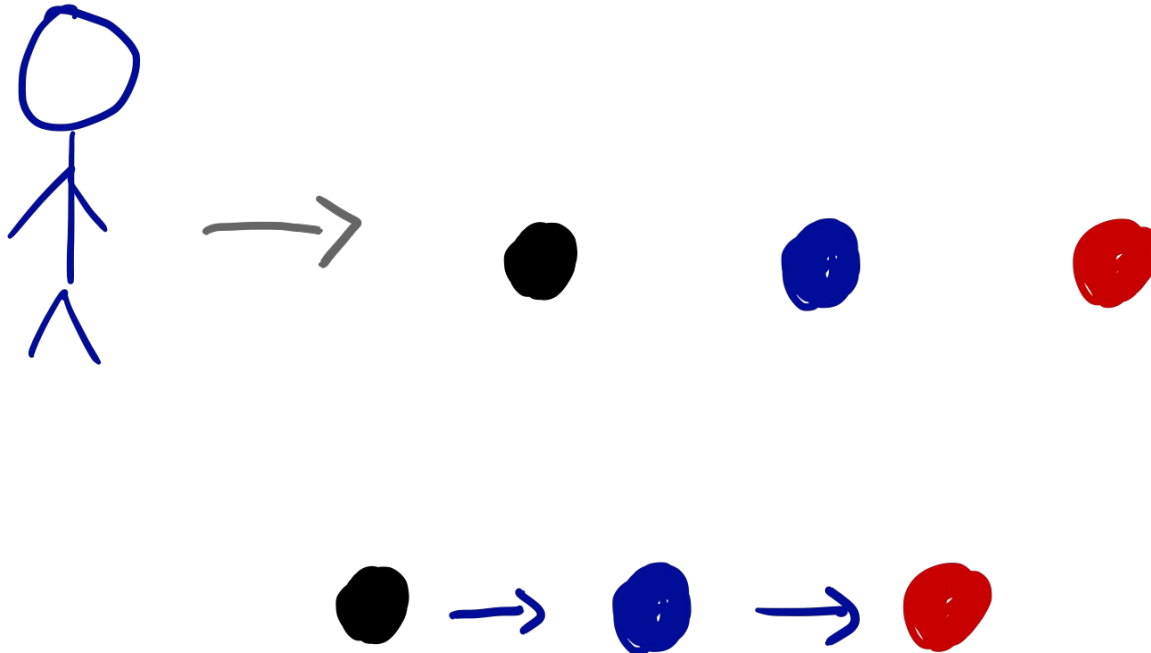
Perché è difficile concettualmente

- In che ordine saranno prese le palline ?



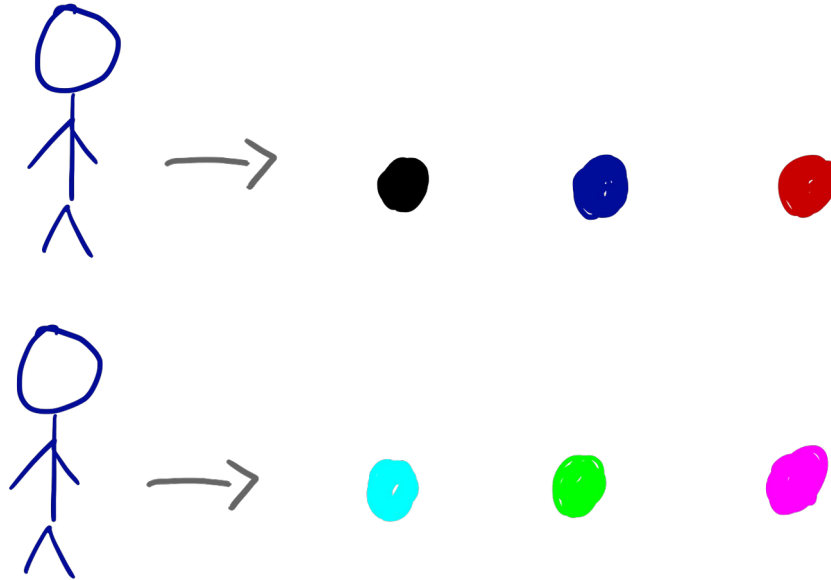
Perché è difficile concettualmente

- In che ordine saranno prese le palline ?



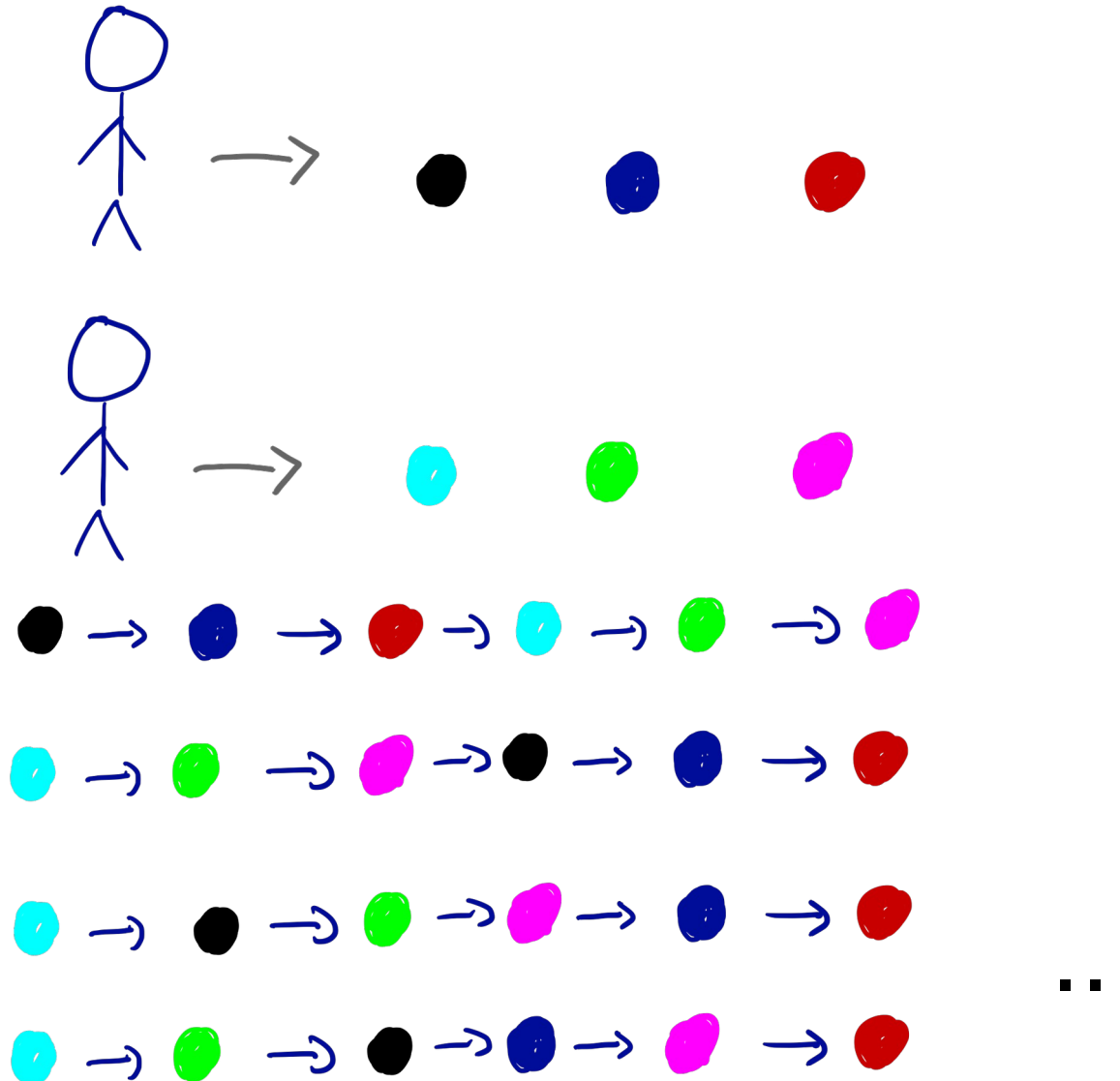
Perché è difficile concettualmente

- In che ordine saranno prese le palline ?



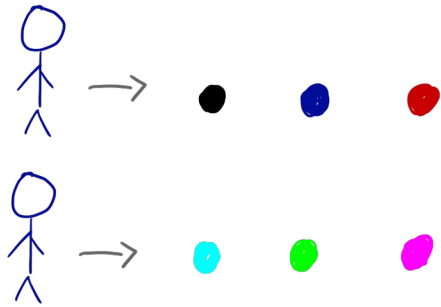
Perché è difficile concettualmente

- In che ordine saranno prese le palline ?

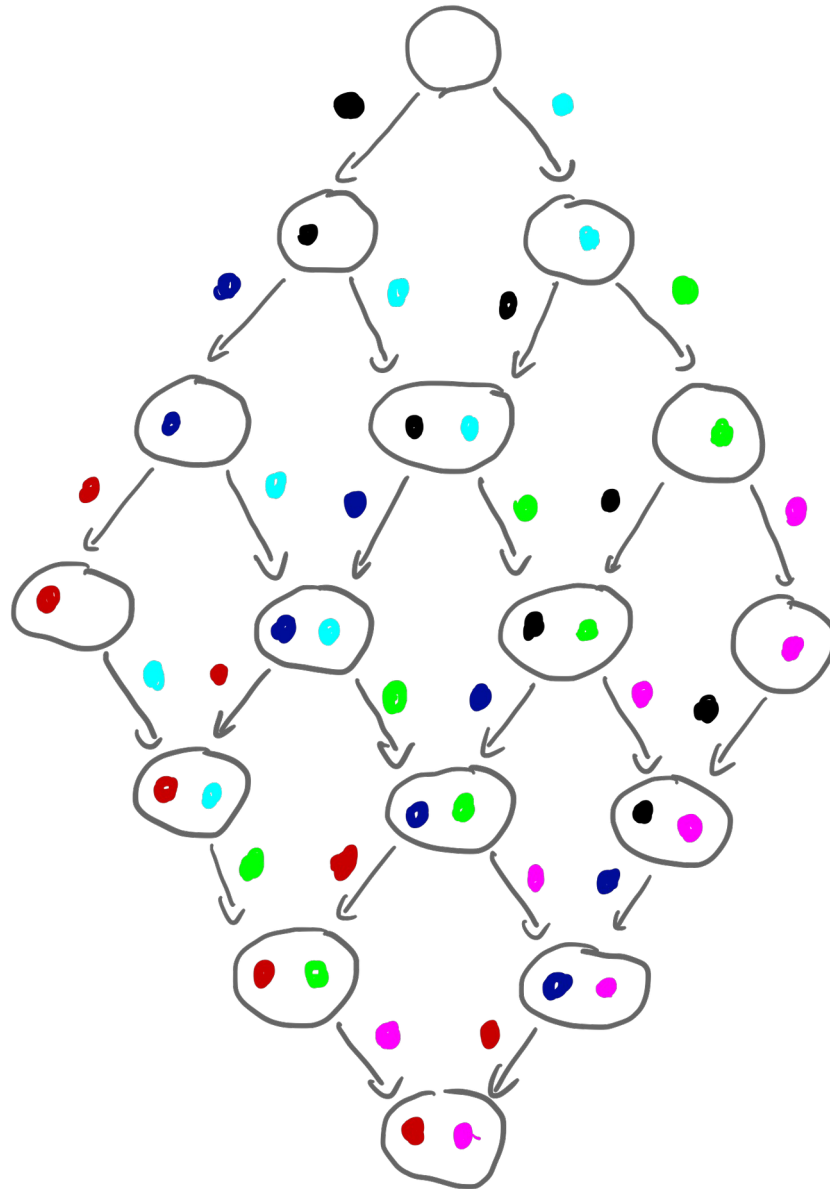


PCAD - INTRODUZIONE

Perché è difficile concettualmente



**Grande
numero di
possibilità**



Perché è difficile concettualmente

- Se ho n uomini e m palline, quante possibilità ?
 - $(n*m)! / (m!)^n$
- Per il nostro esempio, fa : $6! / (3!)^2 = 720/36 = 20$
- Questo numero cresce molto velocemente, per esempio con:
 - 4 palline : $40320 / 576 = 70$
 - 5 palline : $3628800 / 14400 = 252$

Legame con la programmazione

Process P1:

```
int x,y  
a1 : x=10 ;  
a2 : y=20
```

Process P2 :

```
int u,v  
b1 : u=3;  
b2 : v=56  
b3 : u=10 ;
```

Tanti ordini di esecuzione possibili

a1,a2,b1,b2,b3

b1,b2,b3,a1,a2

a1,b1,a2,b2,b3

a1,b1,b2,a2,b3

a1,b1,b2,b3,a3

...

Perché è difficile concettualmente

- Alice ha un gatto e Bob ha un cane
- Condividono un giardino gigante
- Vogliono ciascuno fare uscire il loro animale nel giardino ma mai allo stesso momento
- Devono mettersi d'accordo su quando possono fare uscire il loro animale
 - Problema di **mutua esclusione**
- Non basta guardare il giardino (è troppo grande)
- Urlare dalla finestra o usare il telefonino non funziona, cosa succede se l'altro non risponde ?
- Si mettono d'accordo su un **protocollo** dove ciascuno ha una bandiera alla finestra

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Aspetta che la bandiera di Alice sia bassa
3. Libera il cane
4. Quando il cane torna, abbassa la bandiera

**Problema: cosa succede
se alzano entrambi la loro
bandiera => DEADLOCK**

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Finché la bandiera di Alice è alta :
 - 2.a Abbassa la bandiera
 - 2.b Aspetta che si abbassi la bandiera di Alice
 - 2.c Alza la bandiera
3. Libera il cane
4. Quando il cane torna, abbassa la bandiera

**Questo protocollo garantisce
la mutua esclusione nel
giardino ?**

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Finche la bandiera di Alice è alta :
 - 2.a Abassa la bandiera
 - 2.b Aspetta che si abbassi la bandiera di Alice
 - 2.c Alza la bandiera
3. Libera il cane
4. Quando il cane torna, abbassa la bandiera

Questo protocollo garantisce
la **mutua esclusione** nel
giardino ?
=> SI

Si dimostra via contraddizione

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Finché la bandiera di Alice è alta :
 - 2.a Abbassa la bandiera
 - 2.b Aspetta che si abbassi la bandiera di Alice
 - 2.c Alza la bandiera
3. Libera il cane
4. Quando il cane torna, abbassa la bandiera

Questo protocollo garantisce
l'assenza di **deadlock** nel
giardino ?
=> SI

Si dimostra via contraddizione

Perché è difficile concettualmente

Alice vuole lasciare il gatto :

1. Alza la bandiera
2. Aspetta che la bandiera di Bob sia bassa
3. Libera il gatto
4. Quando il gatto torna, abbassa la bandiera

Bob vuole lasciare il cane :

1. Alza la bandiera
2. Finché la bandiera di Alice è alta :
 - 2.a Abbassa la bandiera
 - 2.b Aspetta che si abbassi la bandiera di Alice
 - 2.c Alza la bandiera
3. Libera il cane
4. Quando il cane torna, abbassa la bandiera

**Questo protocollo garantisce
che sia Alice sia Bob
possono usare il giardino
quando vogliono
=> NO**

**Dipende quando guardano
la bandiera!!!!**

Che cosa è un programma concorrente

- È un insieme di **programmi sequenziali (processi)** che si eseguono in **parallelo** e che **comunicano**
- **Parallelismo:**
 - programmi che si eseguono su processori diversi
 - ma anche programmi multi-task che si eseguono sullo stesso processore (illusione del parallelismo)
- **Comunicazione:**
 - scambio di dati (tramite messaggi o memoria condivisa)
 - sincronizzazione

=> via **memoria condivisa** o via **scambio di messaggi**

Esecuzione concorrente

- Esecuzione di un programma concorrente
 - Insieme di processi (programmi sequenziali)
 - Ciascun processo dispone di un insieme di **istruzioni atomiche**
 - Ogni processo ha un 'puntatore' verso la prossima istruzione da eseguire
 - L'esecuzione del programma concorrente è un **arbitrario intreccio** delle istruzioni dei vari processi (**interleaving**)
 - Durante un' esecuzione, la prossima istruzione è dunque scelta fra quelle puntate dai vari processi

=> Tanti scenari possibili

Comunicazione via memoria condivisa

- I processi hanno un accesso comune alla memoria.
 - Accesso tramite variabili semplici condivise che possono essere **lette** o **scritte**
 - Bisogna stare attenti come è gestito l'accesso a queste variabile (transazione **atomica** o non atomica)
 - Vari casi possibili:
 - Variabili in lettura e scrittura per più processi
=> '**Multiple Reader/Multiple Writer**'
 - Variabili in lettura per più processi ma in scrittura per solo un processo
=> '**Multiple Reader/Single Writer**'
 - Accesso tramite strutture più complesse (semafori, monitor, variabili di condizione, ...)

Esecuzione concorrente

int n=0; //variabile condivisa

Process P1

```
int x;  
a1: x=1;  
a2: n=x;
```

Process P2

```
int u;  
b1: u=2;  
b2: n=u;
```

