Caching
para hordas y estampidas

Claudio Daniel Freire

^ *soy yo*

Caching

- **Tengo una función costosa pero con resultados estables**
  - **No cambia muy seguido**
  - **No importa que refleje datos algo viejos de vez en cuando**

Caching

Simple:

```
def muy_costoso(x, _cache={}):
    if x not in _cache:
        # muchos cálculos
        _cache[x] = rv = blabla
    return _cache[x]
```

Caching

Listo

The end


NO


NO
NO

NO

NO

y

NO

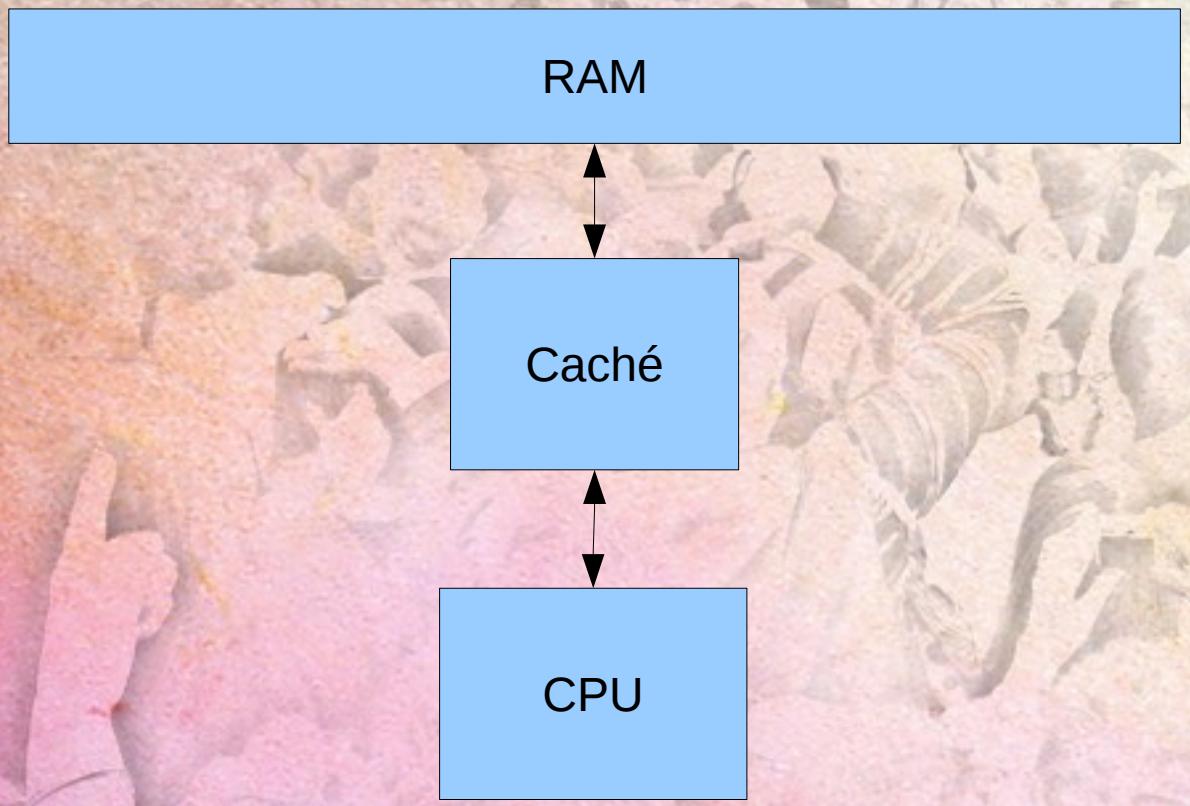
Caching
para cuando importa

Caching
para cuando importa

- **Políticas**
  - **Evicción**
  - **Expiración**
  - **Limitación**
- **Arquitecturas más complejas**
- **Problemas de escala**
  - **Coherencia**
  - **Concurrencia**

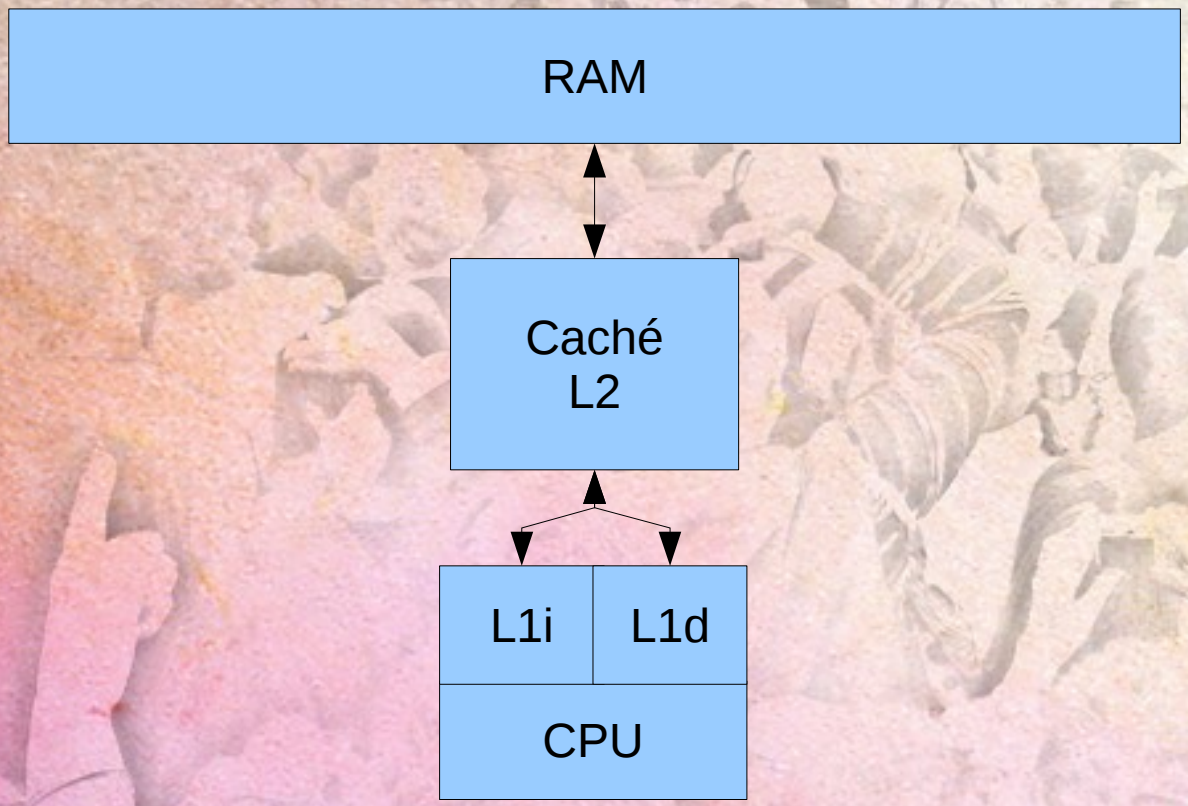
Caching
en los procesadores

Simple simplísimo simple



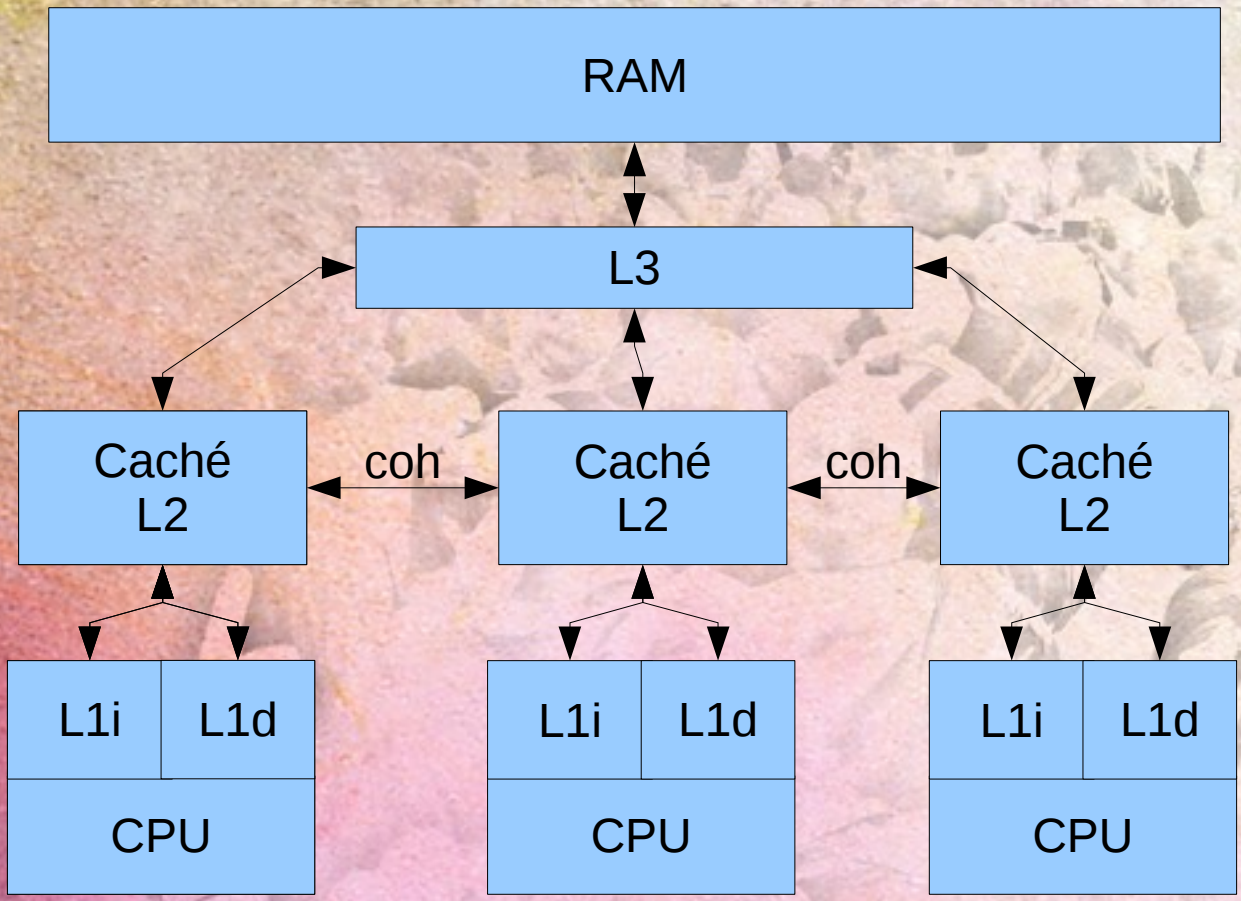
Caching
en los procesadores

Complicala un poco



Caching
en los procesadores

Faso loco Faso



Caching
políticas

Con arquitecturas complejas...

...los detalles importan

Caching
políticas

- **Qué y por cuánto tiempo**
- **Utilización de recursos**
- **Relevancia de los datos**

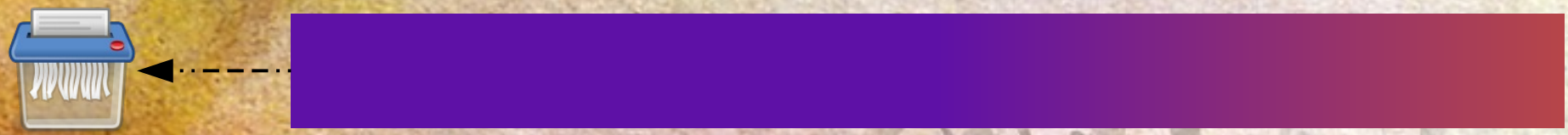
Caching
políticas

- **Qué y por cuánto tiempo**
- **Utilización de recursos**
- **Relevancia de los datos**
- **Correctitud de los datos**
  - **coherencia**

Caching
políticas – evicción

- **LRU**
- **LFU**

Caching
políticas – evicción



Items Fríos

Items Dudosos

Items Calientes

Caching
políticas – evicción

Diccionario

- **Importantísimo controlar el orden de salida de los datos**
  - **Asegurarse de quitar los FRÍOS**
- **Importantísimo SACAR**
  - **Evitar el crecimiento infinito del caché**

Caching
políticas – evicción

Sin uso
hace tiempo

Recientemente
Usados

- LRU supone que el *working set* entra en el caché
  - Luego, al llenarse, los datos menos recientemente usados están fuera del *working set* y probablemente no sean necesarios en un futuro cercano
- FIFO como una simplificación

Caching
políticas – evicción

Infrecuentemente usados

Frecuentemente Usados

- **LFU supone un costo de cómputo uniforme, y un working set grande**
  - **Luego, quitar los menos frecuentemente usados, minimiza el costo de recómputo para un tamaño dado**

Caching
políticas – evicción

Infrecuentemente usados

LRU

LRU

LRU

LRU

Frecuentemente Usados

- **MQ**
  - **Usar muchas LRU en forma LFU**
  - **http://static.usenix.org/event/usenix01/zhou.html**

Caching
políticas – evicción

- **MQ – à la memcache**
  - **Usar muchas LRU por tamaño**
  - **Asume diferente tamaño ~ diferente propósito ~ diferente costo de recómputo**
  - **Funciona**

Caching
políticas – limitación

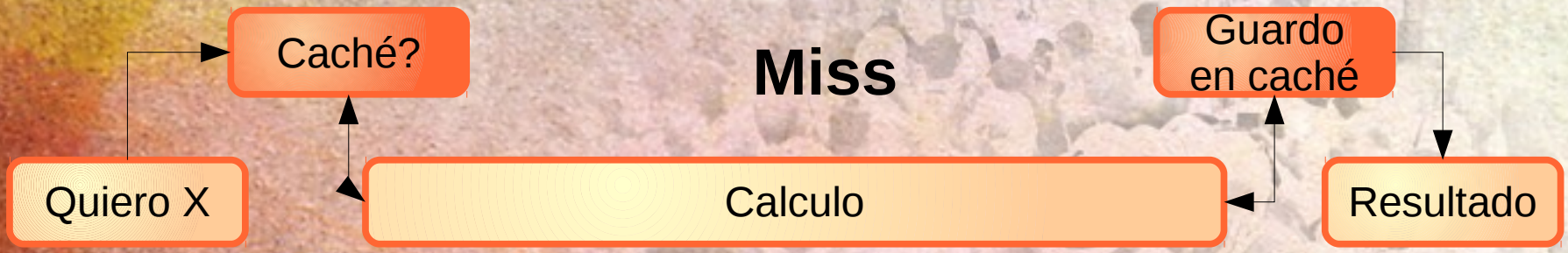
- **Entradas**
  - **Hasta *N* resultados sin importar tamaño**
  - **Sirve cuando son todos similares**
- **Bytes**
  - **Hasta *M* bytes máximo**
  - **Sirve para datos de diversos tamaños**
  - **Fuerte garantía de uso de recursos**
- **Tiempo**
  - **Por a lo sumo *T* tiempo (suele combinarse)**

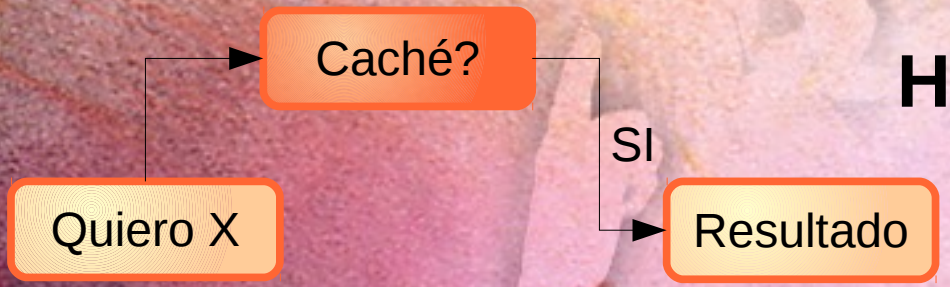
Caching
políticas – refresco

- **Sincrónico**
  - **Clásico, simple, siempre correcto**
- **Dogpile**
  - **Transparente, escalable, en especial con cómputos que llevan tiempo, a expensas de una pizca de correctitud**
- **Preemptive**
  - **Una cereza agregable a las de arriba**
  - **dogpile + cereza = cake(walk)**

Caching
políticas – refresco

Sincrónico: busco, no está, mala chuerte, calculo

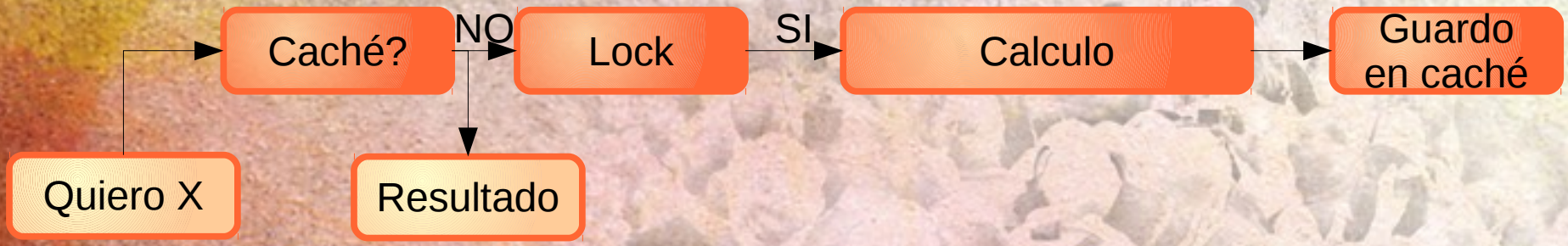




Caching
políticas – refresco

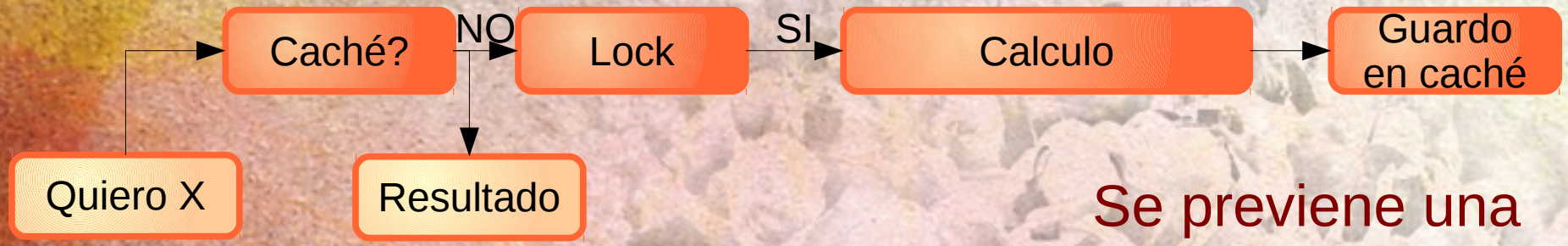
Dogpile (a grandes rasgos)
uso el viejo *mientras calculo*

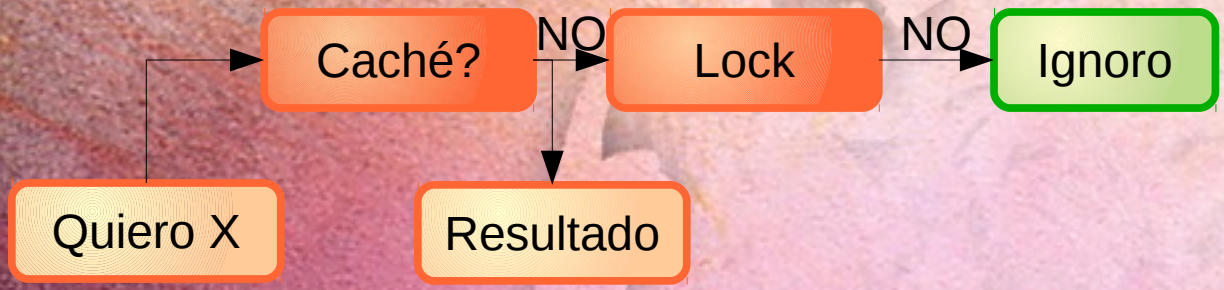
Miss



Caching
políticas – refresco

Miss





Se previene una literal fork-bomb de recálculos en paralelo con un lock, llamado... adivinen...

DOGPILE

Caching
políticas – refresco

Preemptive

- Aprovecho tiempos muertos para calcular
  - Ej: A la noche, recalculo el homepage
- Aprovecho resultados relacionados
  - numMessages(), getMessages()
  - getMessages() actualiza el caché de numMessages()
  - También mejora la coherencia (wiii)

Caching
políticas

El Diccionario es el DEMONIO

- No tiene políticas, ni las va a tener
  - La sintaxis de diccionario no soporta TTL
- Crece indefinidamente
- No es eficiente controlar qué se quita cuando se limpia manualmente
- No es eficiente controlarlo. Punto.

Caching – pasando a web

arquitecturas

Caching – pasando a web
arquitecturas

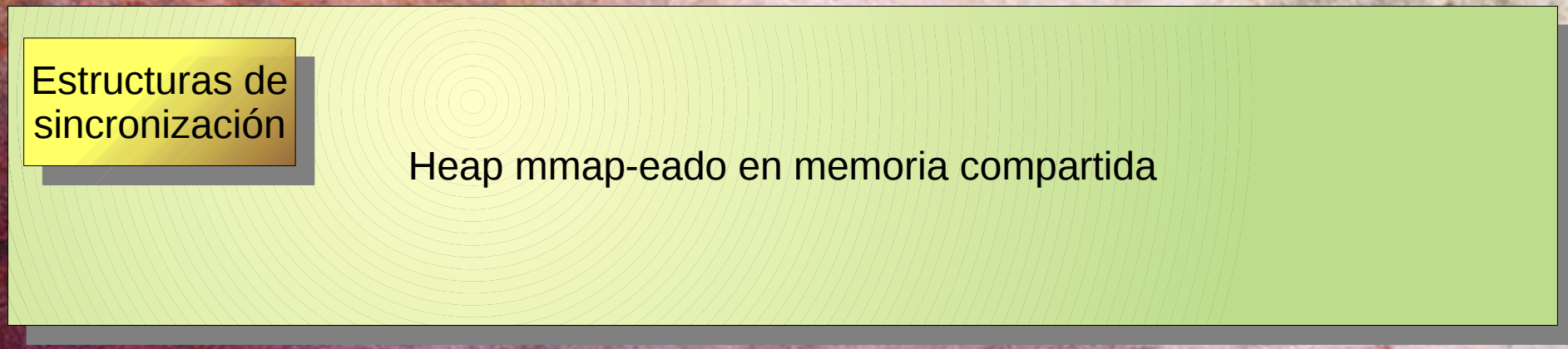
- **Inproc**
  - **Estructuras eficientes para guardar referencias dentro del proceso**
- **Memoria compartida**
  - **Técnicas para compartir instancias entre procesos (sin serializar)**
- **Externos**
  - **Serialización hacia procesos externos**
    - **memcached, bases de datos**

Caching – pasando a web
arquitecturas

- **Inproc**
  - **LRU**
    - **Heap**
    - **FIFO**

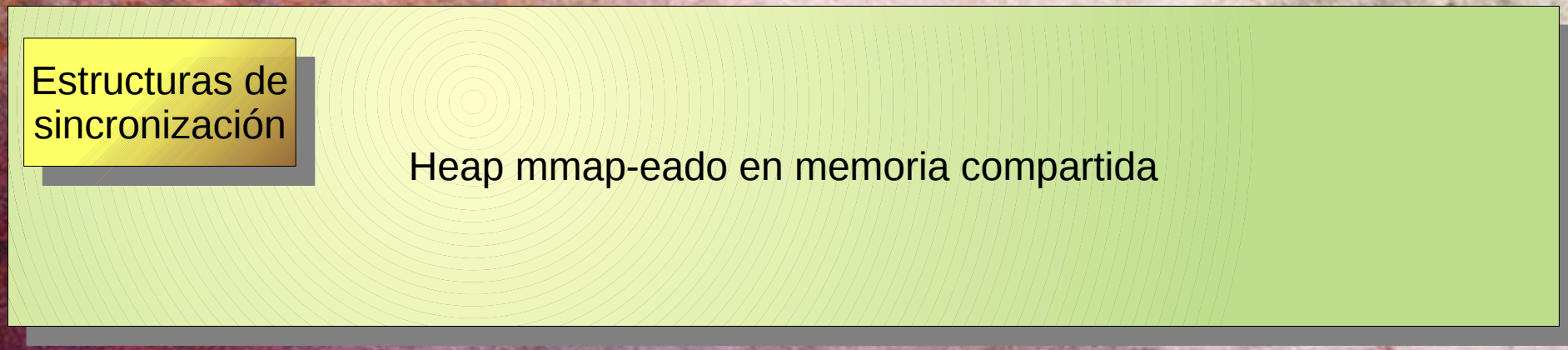
Caching – pasando a web
arquitecturas

- **Memoria compartida**
  - **proxies + mmap**
  - **Muy eficiente, pero de implementación dolorosa**
    - **Hay que implementar casi todo de cero**
    - **multiprocessing ayuda a sincronizar**



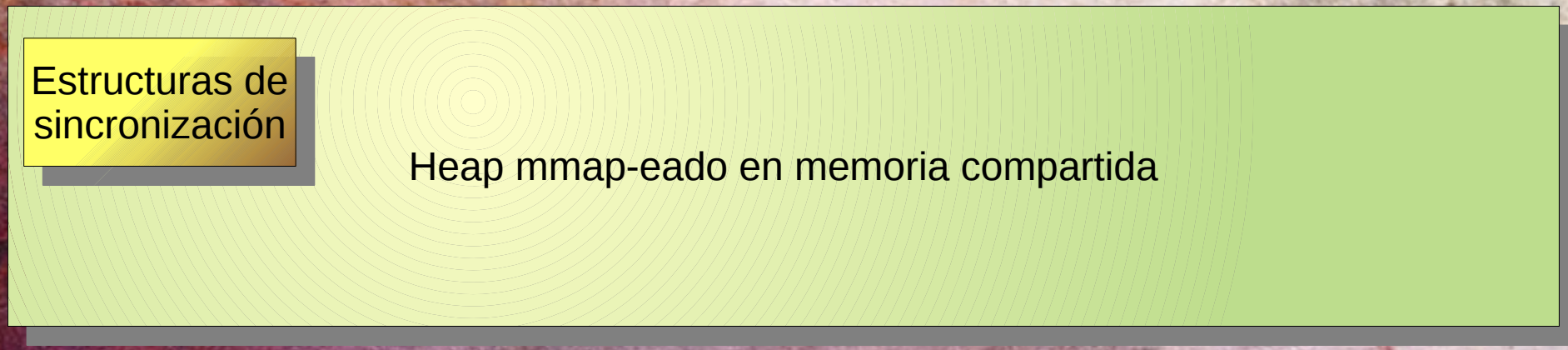
Caching – pasando a web
arquitecturas

- **Memoria compartida – bueno para**
  - **byte strings**
  - **arrays numéricos *(numpy)***



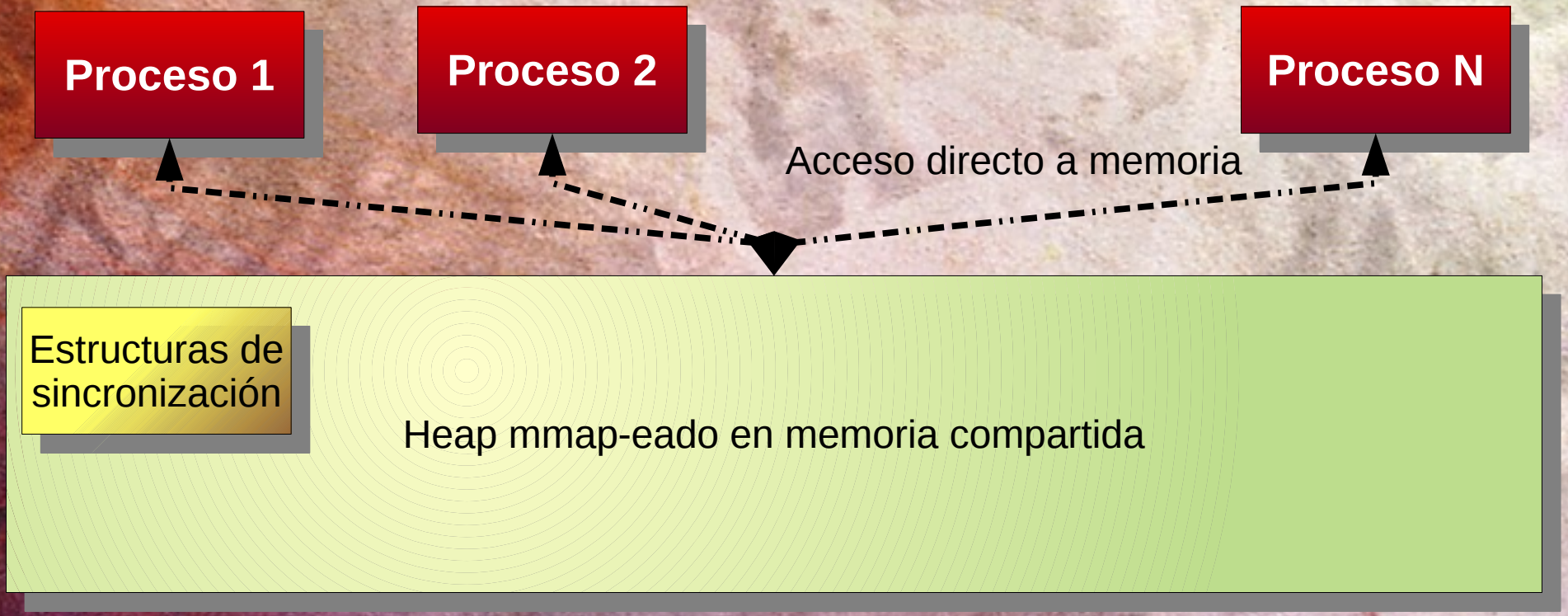
Caching – pasando a web
arquitecturas

- **Memoria compartida – OJO**
  - **locking granular (sino va a haber contención)**
  - **no serializar/deserializar**
    - **para eso usar externos**
  - **estructuras complejas requieren ayuda de C**
    - **o cython**



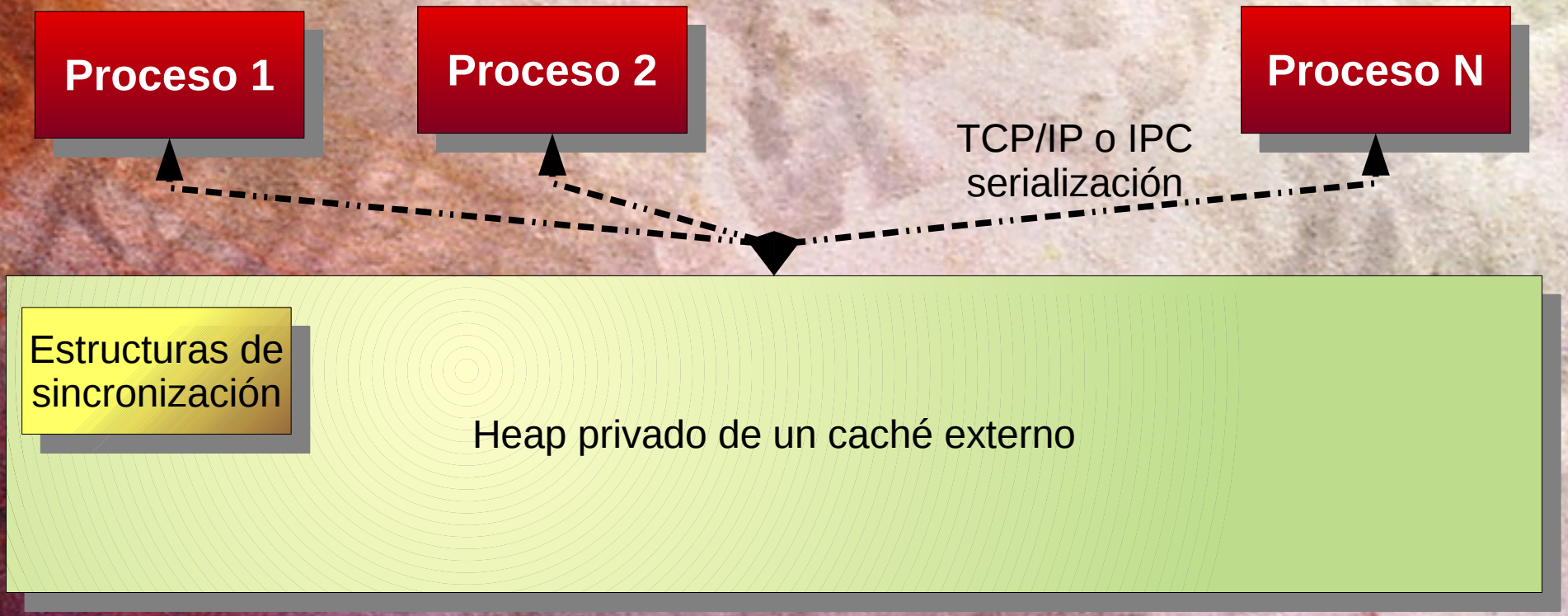
Caching – pasando a web
arquitecturas

- **Memoria compartida**



Caching – pasando a web
arquitecturas

- **Externos**



Caching – pasando a web
arquitecturas

- **Externos**
  - **Mayor flexibilidad**
    - **Mover a otro nodo con más memoria**
    - **Compartir caches entre nodos**
    - **Configuración centralizada**
  - **A cambio del overhead de serialización:**
    - **Actúan más como una base de datos rápida y especializada que como un caché in-proc**
    - **Los accesos repetidos son ineficientes, mucho tráfico de red/IPC, mucha serialización**

Caching – pasando a web

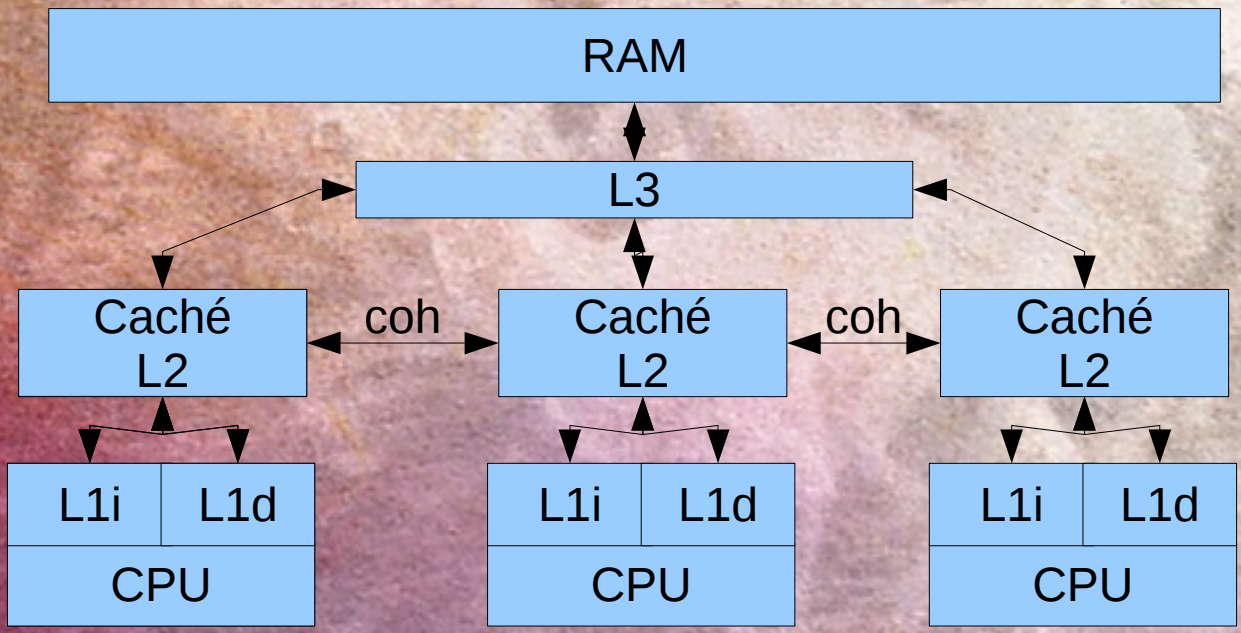
arquitecturas
piedra papel Y tijera

- **Tiers**
  - **También llamados niveles**
    - **O capas**

Caching – pasando a web

arquitecturas
piedra papel Y tijera

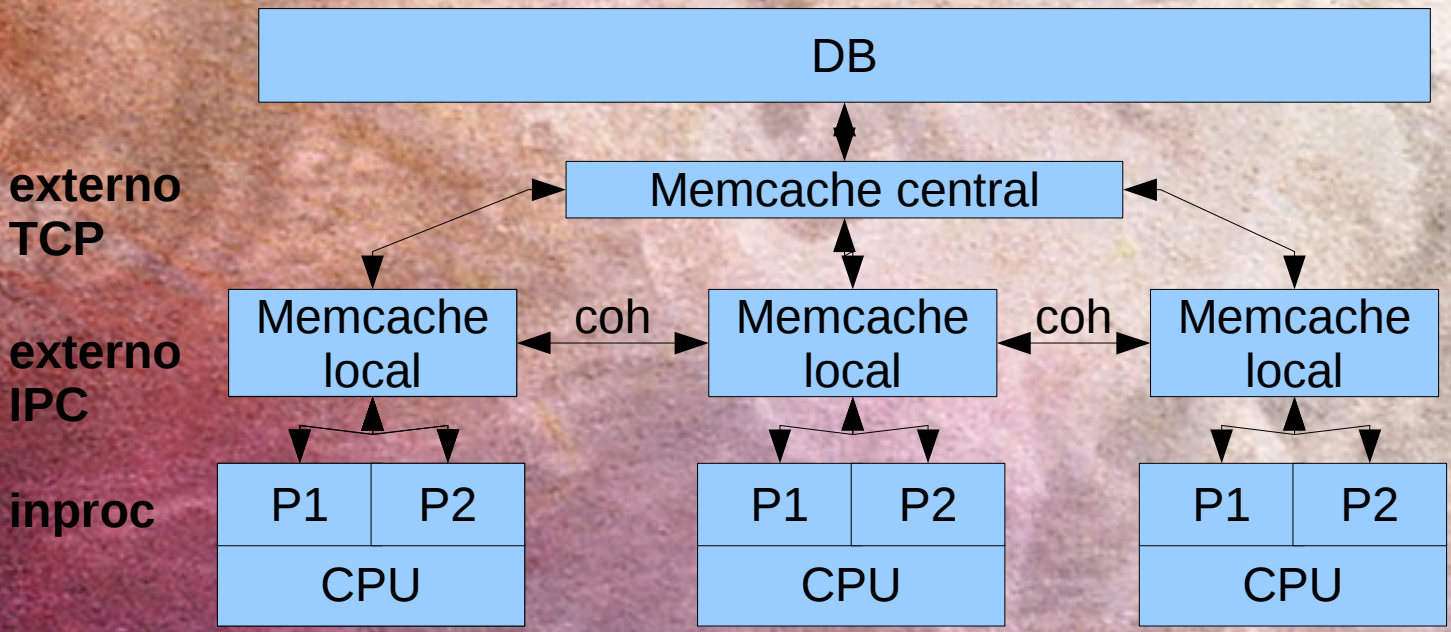
- **Tiers**
  - **También llamados niveles**
    - **O capas**



Caching – pasando a web

arquitecturas
piedra papel Y tijera

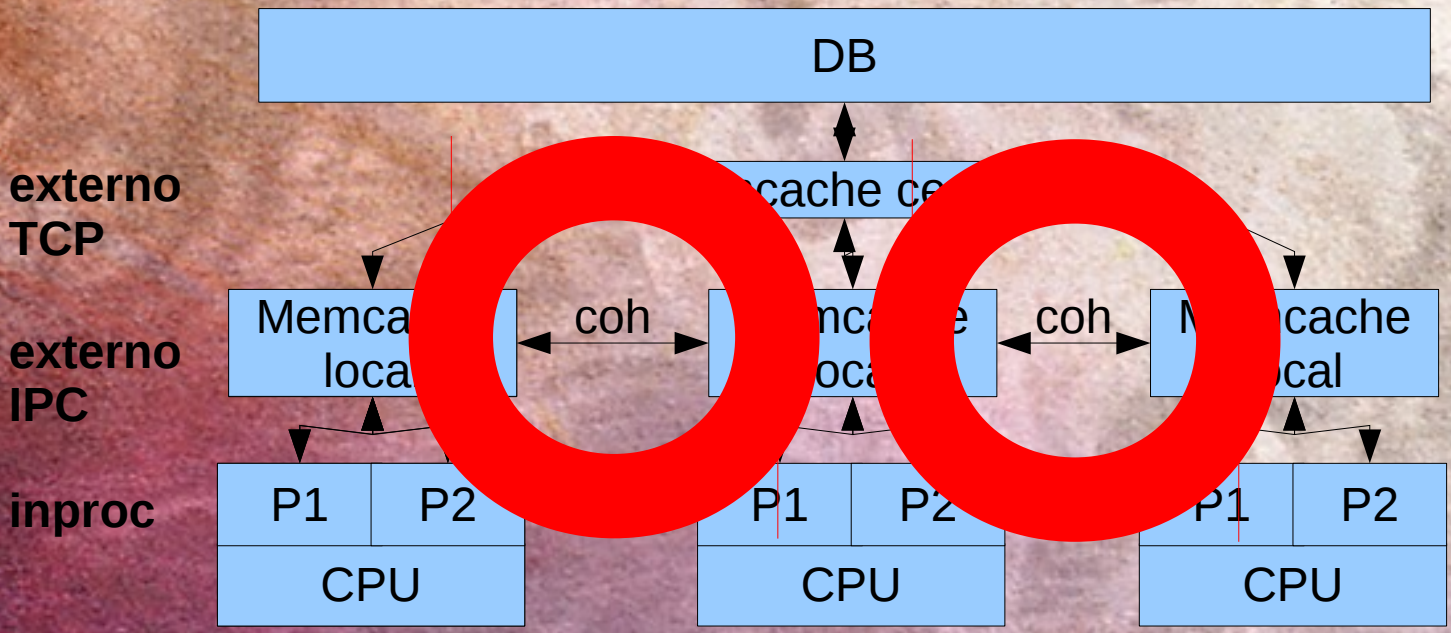
- **Tiers**
  - **También llamados niveles**
    - **O capas**



Caching – pasando a web

arquitecturas
piedra papel Y tijera

- **Tiers**
  - **También llamados niveles**
    - **O capas**



Caching – pasando a web

arquitecturas
piedra papel Y tijera

- **Tiers**
  - **Tienen sus vueltas**
    - **Inclusivos vs exclusivos**
    - **Políticas de promoción**
    - **Políticas de escritura**
      - **write-through**
      - **write-back**
  - **Complejos complejísimos**
    - **A veces se vuelve difícil predecir su comportamiento**

Caching – pasando a web

arquitecturas piedra papel Y tijera

- **Tiers**
  - **Los niveles más altos manejan cargas exhorbitantes**
    - **Sharding**
      - **por zonas**
      - **hashing consistente**
    - **Replicación**
      - **manual (escribir en ambos, leer round-robin)**
      - **automático (algunos motores lo soportan)**

Caching – pasando a web
coherencia

- **Tengo dos funciones relacionadas**
  - **Que sus datos sean coherentes entre sí**
- **Tengo dos nodos que pueden escribir**
  - **Que el usuario no vea datos cacheados viejos**

Caching – pasando a web

coherencia

```
@cached
def getMessages(userId):
    …
    return X


@cached
def numMessages(userId):
    return len(getMessages)
```

Caching – pasando a web
coherencia

- **No cachear**
  - **que numMessages compute siempre, basándose en el resultado cacheado de getMessages**
  - **Funciona si el cómputo no es costoso**
- **Invalidar oportunísticamente**
- **Escribir oportunísticamente**

Caching – pasando a web
coherencia

- **No cachear**
- **Invalidar oportunísticamente**
  - **getMessages() invalida numMessages()**
  - **numMessages() escucha getMessages()**
- **Escribir oportunísticamente**

Caching – pasando a web
coherencia

```
@cached
def getMessages(userId):
    …
    numMessages.invalidate(userId)
    return X
@cached
@getMessages.invalidates
def numMessages(userId):
    return len(getMessages)
```

Caching – pasando a web
coherencia

- **No cachear**
- **Invalidar oportunísticamente**
- **Escribir oportunísticamente**
  - **getMessages() actualiza numMessages()**

Caching – pasando a web
coherencia

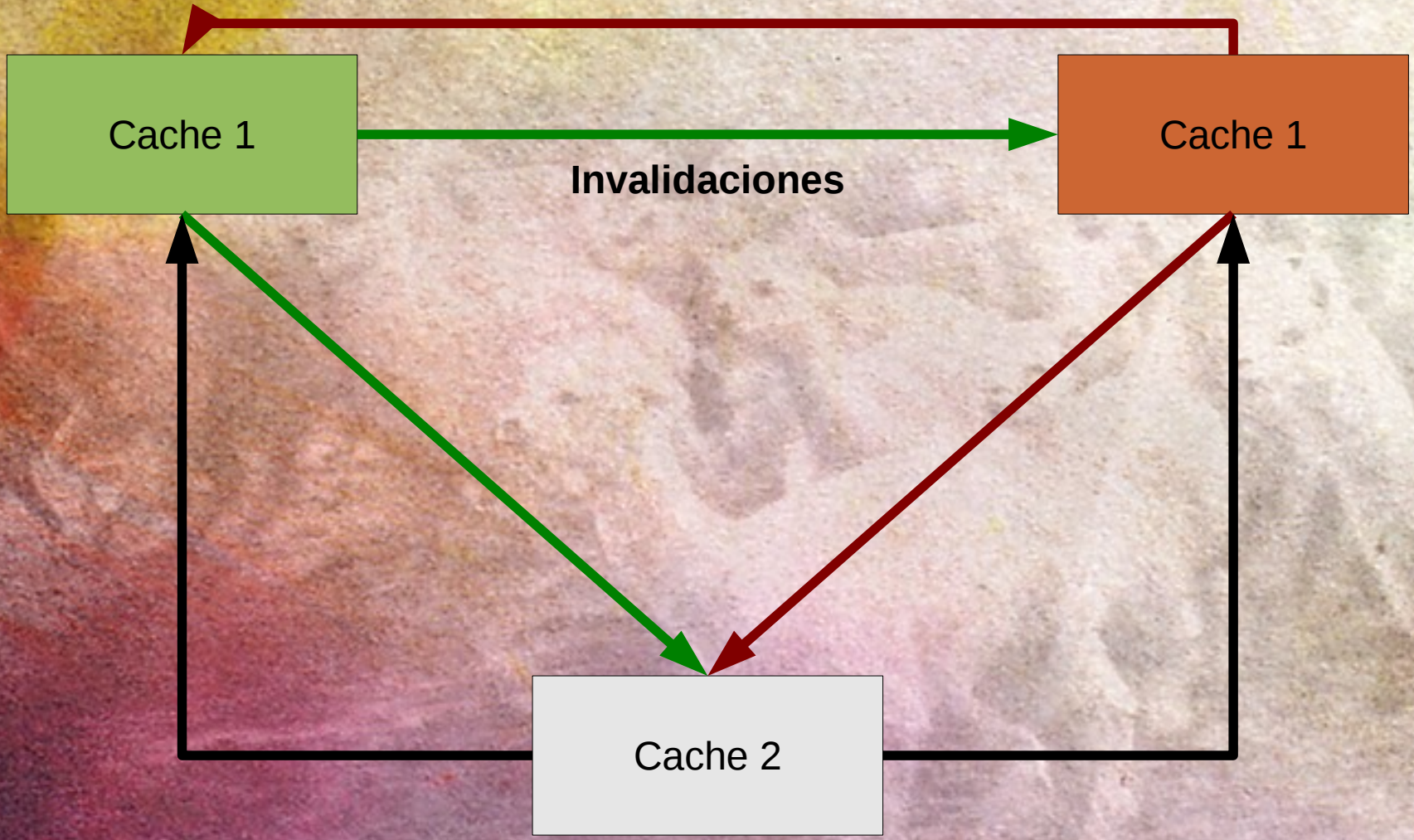
```
@cached
def getMessages(userId):
    …
    numMessages.put(len(X), userId)
    return X

@cached
def numMessages(userId):
    return len(getMessages)
```

Caching – pasando a web
coherencia

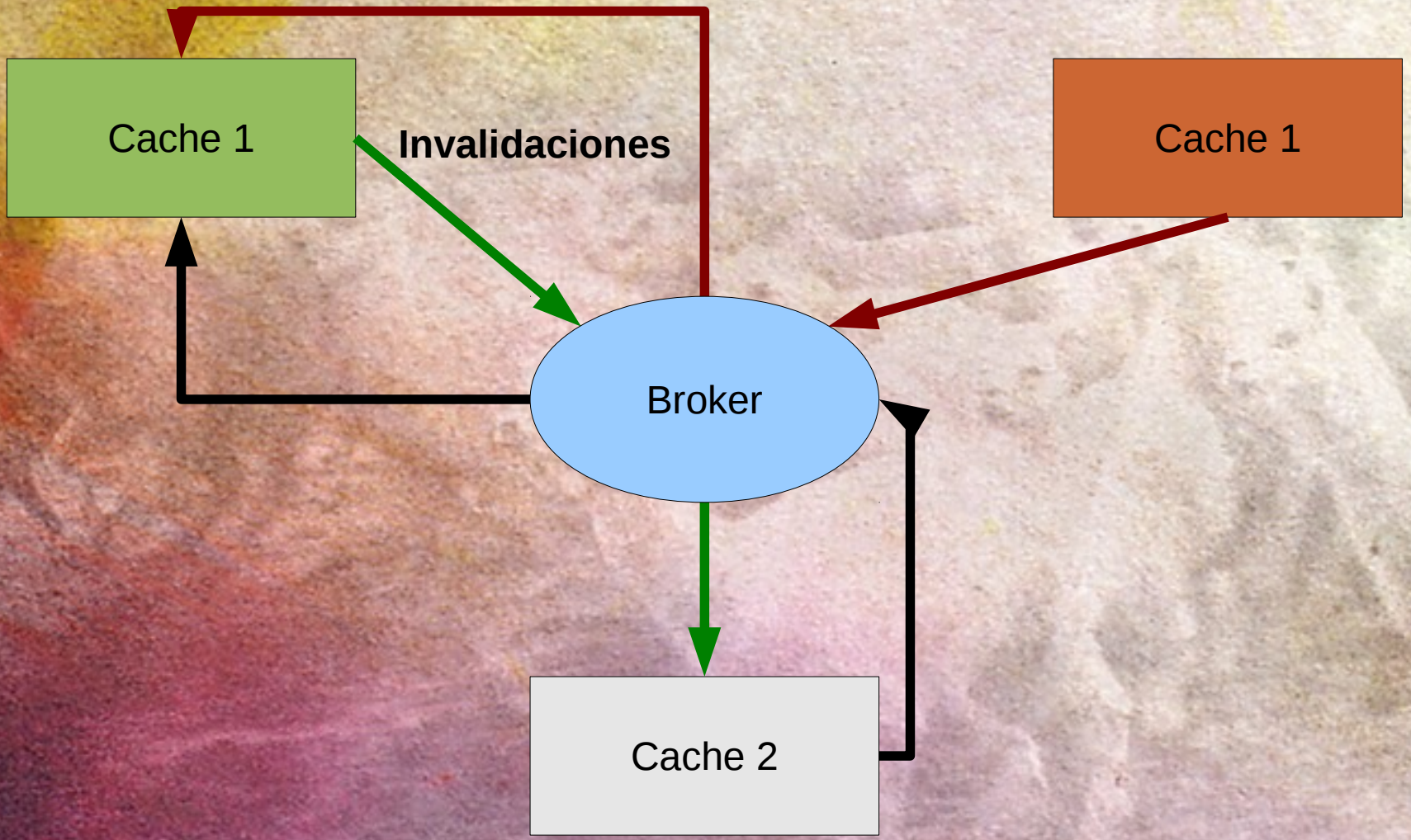
- **Varios caches tienen los mismos datos coordino con mensajes**
  - **Notificar sólo invalidaciones**
  - **Usar un nivel superior para intercambiar datos actualizados**
    - **O comunicación P2P si es ya un nivel superior** *(complejo complejísimo)*

Caching – pasando a web

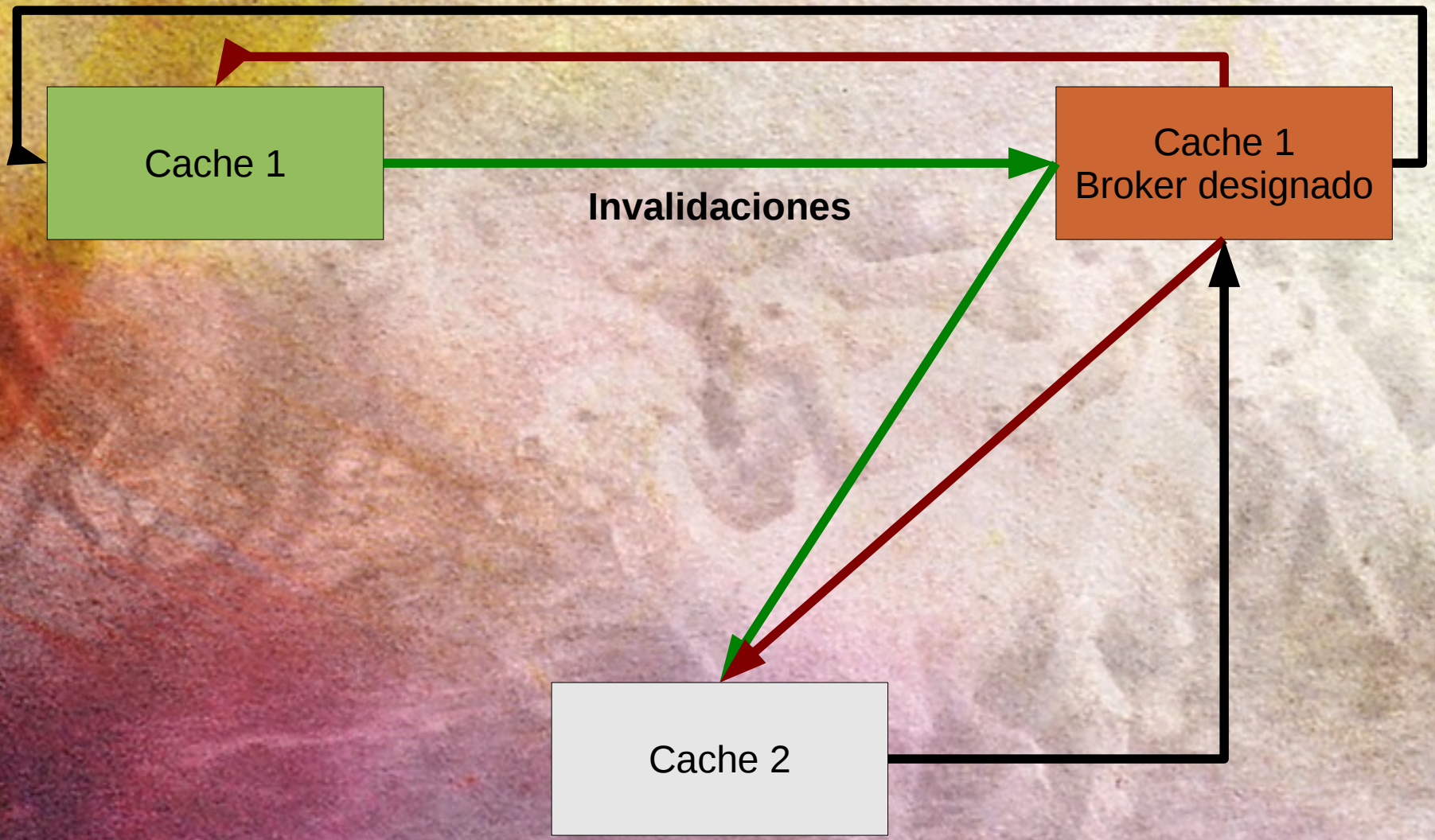
coherencia



Caching – pasando a web
coherencia

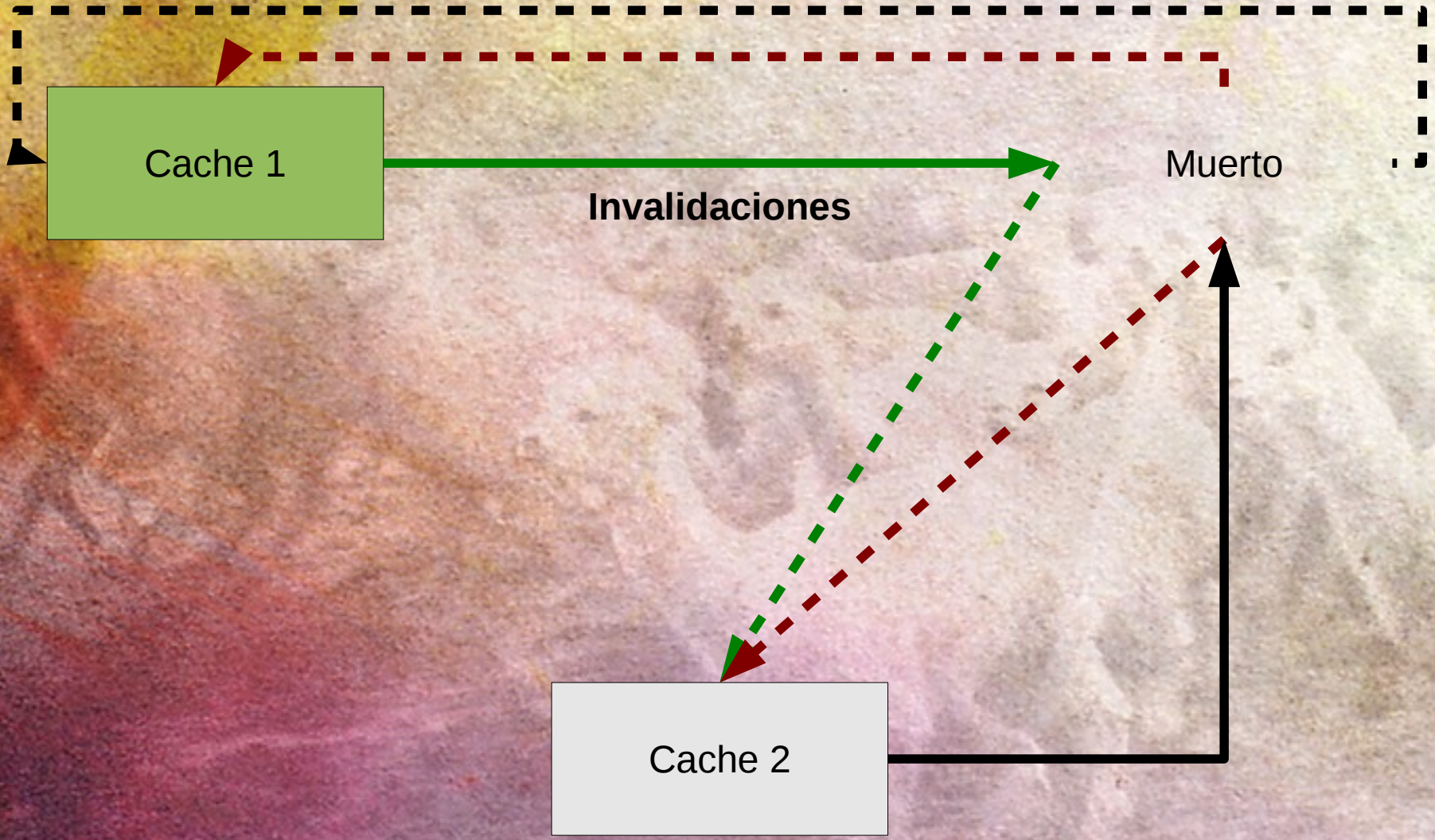


Caching – pasando a web
coherencia

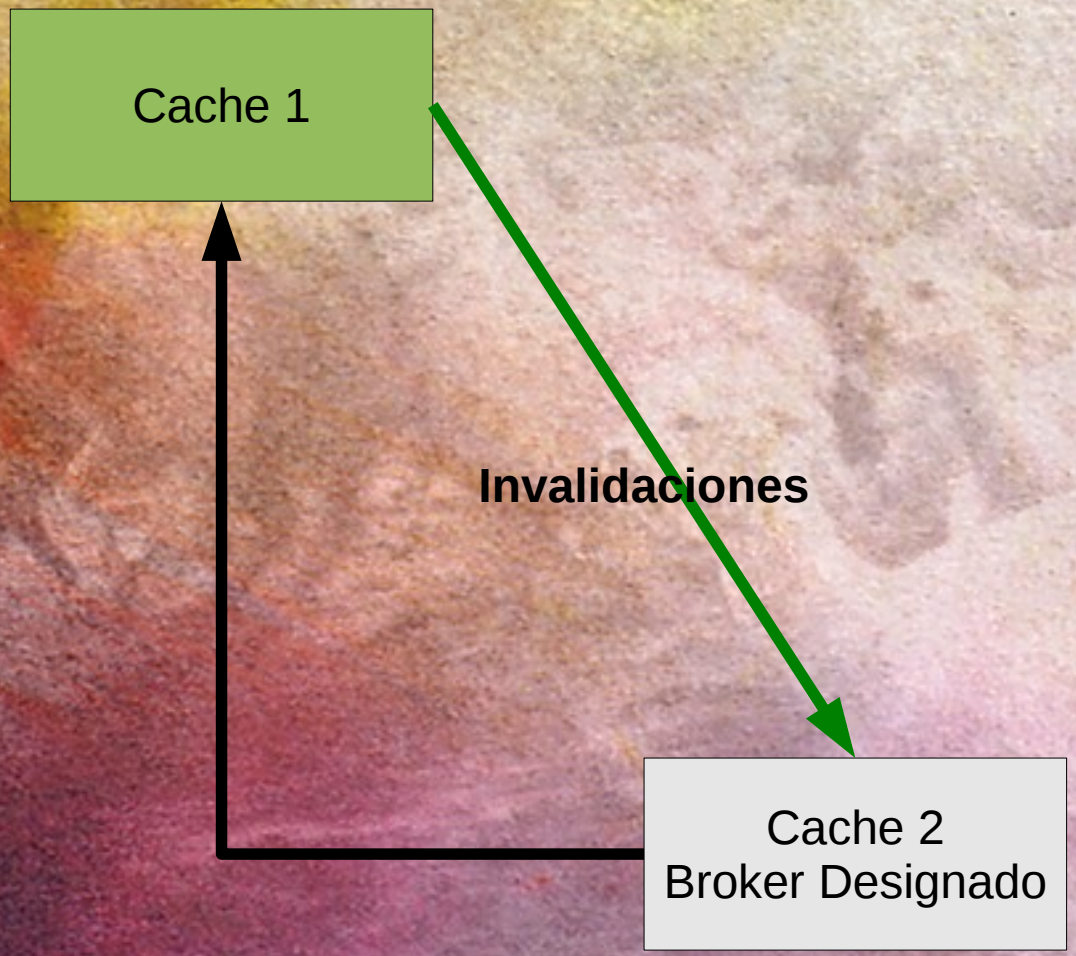


Caching – pasando a web

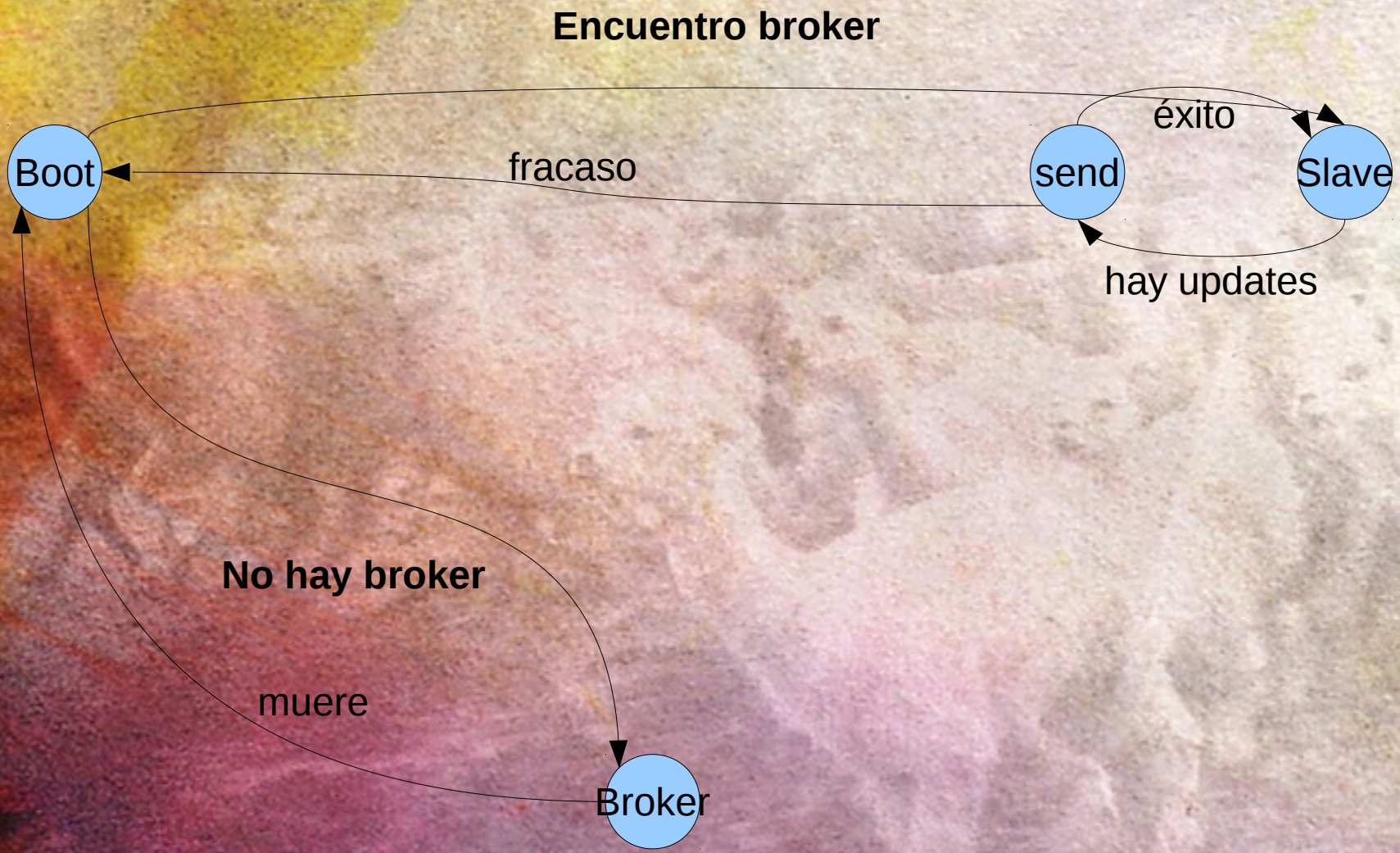
coherencia



Caching – pasando a web
coherencia



Caching – pasando a web
coherencia



Caching – pasando a web
concurrencia

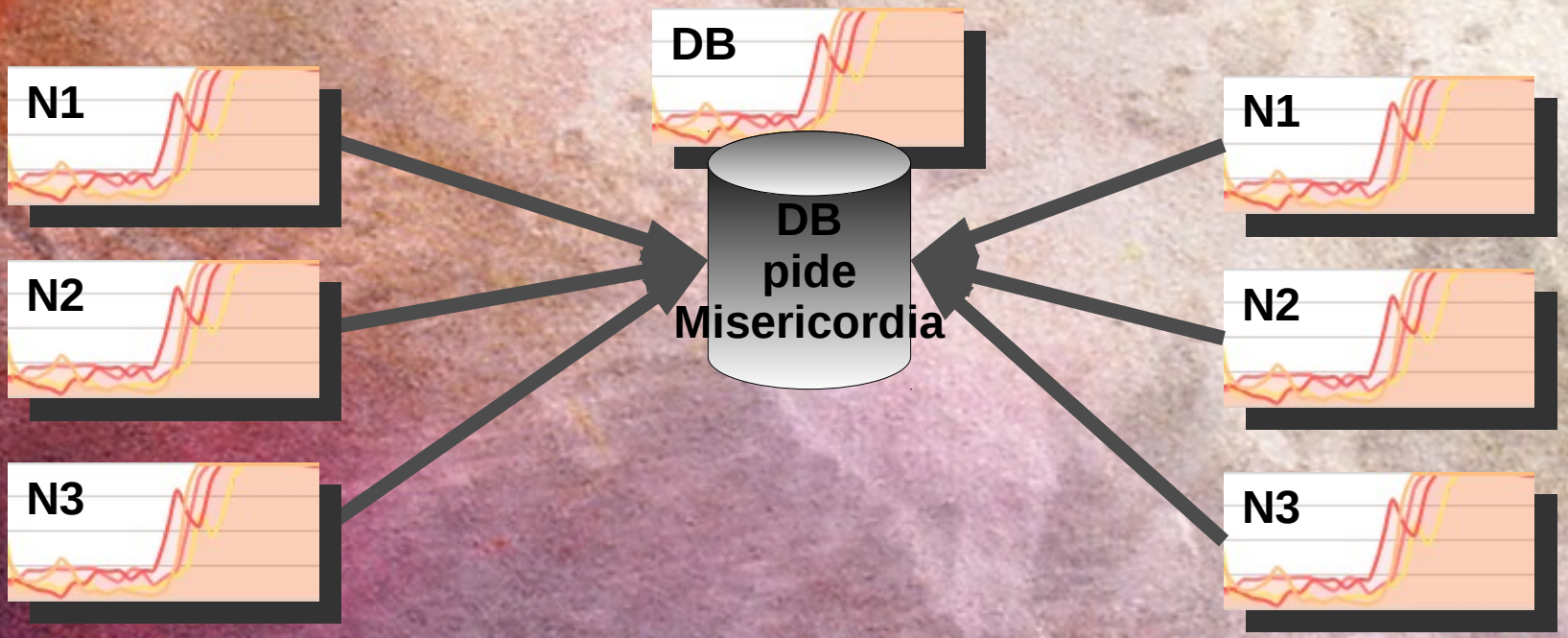
- **Tengo una función muy costosa**
- **Tengo N nodos que la usan**
  - **Quién la calcula cuando expira**

Caching – pasando a web
concurrencia

- **Tengo una función muy costosa**
- **Tengo N nodos que la usan**
  - **Quién la calcula cuando expira**
  - **Alguien**
    - **Sólo 1**
    - **No más de 1**
    - **No MENOS de 1**

Caching – pasando a web
concurrencia

- **Todos calculan**
  - **Stampeding herd**



Caching – pasando a web
concurrencia

- **Dogpile**
  - **Al inicio del cálculo, marco en el cache la intención de calcular**
    - **Ej: con un valor especial**
    - **Los demás esperan**
  - **Problemas:**
    - **Si el que calcula muere?**
      - **dogpile = A lo sumo 1 calcula**
      - **necesito exactamente 1 calcula**

Caching – pasando a web
concurrencia

- **Dogpile**
  - **Problemas:**
    - **Y si no tengo un caché centralizado?**
    - **Y si tengo más de 1 caché?**
      Ej:
      - **Deadlocks!**
  - **O sea:**
    - **no escala**

Caching – pasando a web
concurrencia

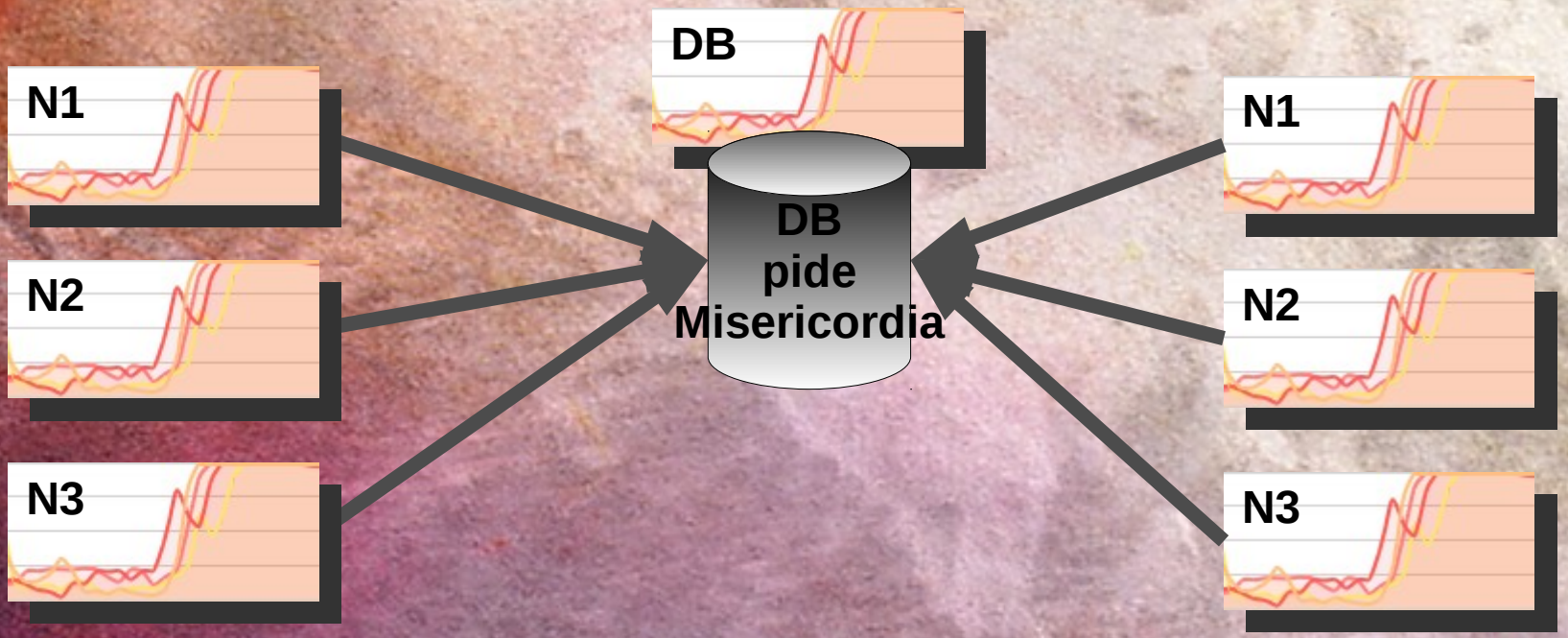
- **Mensajería**
  - **Más complejo**
  - **Pero más flexible**
  - **Más robusto**
  - **En teoría, permite evitar deadlocks**
    - **En la práctica, nunca nos tomamos el trabajo**

Caching – pasando a web
concurrencia

- **Tengo 2 x 10^6 elementos en mi caché**
- **Expiran todos al mismo tiempo**

Caching – pasando a web
concurrencia

- **Todos calculan**
  - **Stampeding herd**



Caching – pasando a web
concurrencia

- **Tengo 2 x 10^6 elementos en mi caché**
- **Expiran todos al mismo tiempo**
- **Problema: son elementos distintos**
  - **No puedo evitar calcularlos**
  - **Es trabajo que *hay que* hacer**
  - **El tema es... cuándo**

Caching – pasando a web
concurrencia

- **Varias tácticas**
  - **Inproc**
    - **Colas de trabajo**
      - **Limitar el trabajo concurrente de cada nodo**
      - **Garantiza una performance estable**
      - **Puede retrasar el cálculo de algunos elementos bajo carga**
    - **Semáforos**
      - **Más pesado que las colas de trabajo**
      - **Hay que cuidarse de deadlocks**
      - **Más sencillos de agregar a una aplicación existente**

Caching – pasando a web
concurrencia

- **Varias tácticas**
  - **Externos**
    - **Colas de trabajo**
      - **Celery**
      - **Custom con ZMQ**
    - **Semáforos**
      - **memcache: incr/decr/gets/cas**
      - **paxos**

Caching – pasando a web
concurrencia

- **Tradeoffs**
  - **Colas de trabajo**
    - **Muchas implementaciones disponibles**
    - **Conceptualmente sencillas**
    - **Fáciles de administrar**
    - **Externas requieren serializar tareas**
    - **Punto de falla**
  - **Semáforos**
    - **Altamente decentralizado y escalable**
    - **Muy propenso a deadlock y amigos**
    - **Pocas implementaciones disponibles**

Caching – pasando a web
concurrencia

- **Magic bullet (bue, casi)**
  - **Bundling**
    - **Calcular varios valores al mismo tiempo**
  - **Ej:**
    - **Obtener mensajes de X**
      → obtener mensajes de X1...Xn
    - **Calcular ratings de Y**
      → calcular ratings de Y1…Yn

Caching – pasando a web
concurrencia

- **Magic bullet (bue, casi)**
  - **Muchas tareas se pueden ACELERAR con bundling**
  - **Netamente menos trabajo**
  - **Caching genera oportunidades de bundling que no se verían sin caching.**
    - **¿por qué uso tanto spanglish?**

Caching
moraleja

- **Arquitecturas**
  - **considerarlas, pensarlas, diseñarlas**
- **Coherencia y concurrencia**
  - **ponderarlas desde el principio, definen arquitecturas, no son plug-ins**
- **Reusar**
  - **muchos problemas complejos, no reinventar la rueda, reusar la rueda de los demás**
- **Bundling**
  - **Se va a dar, aprovecharlo**