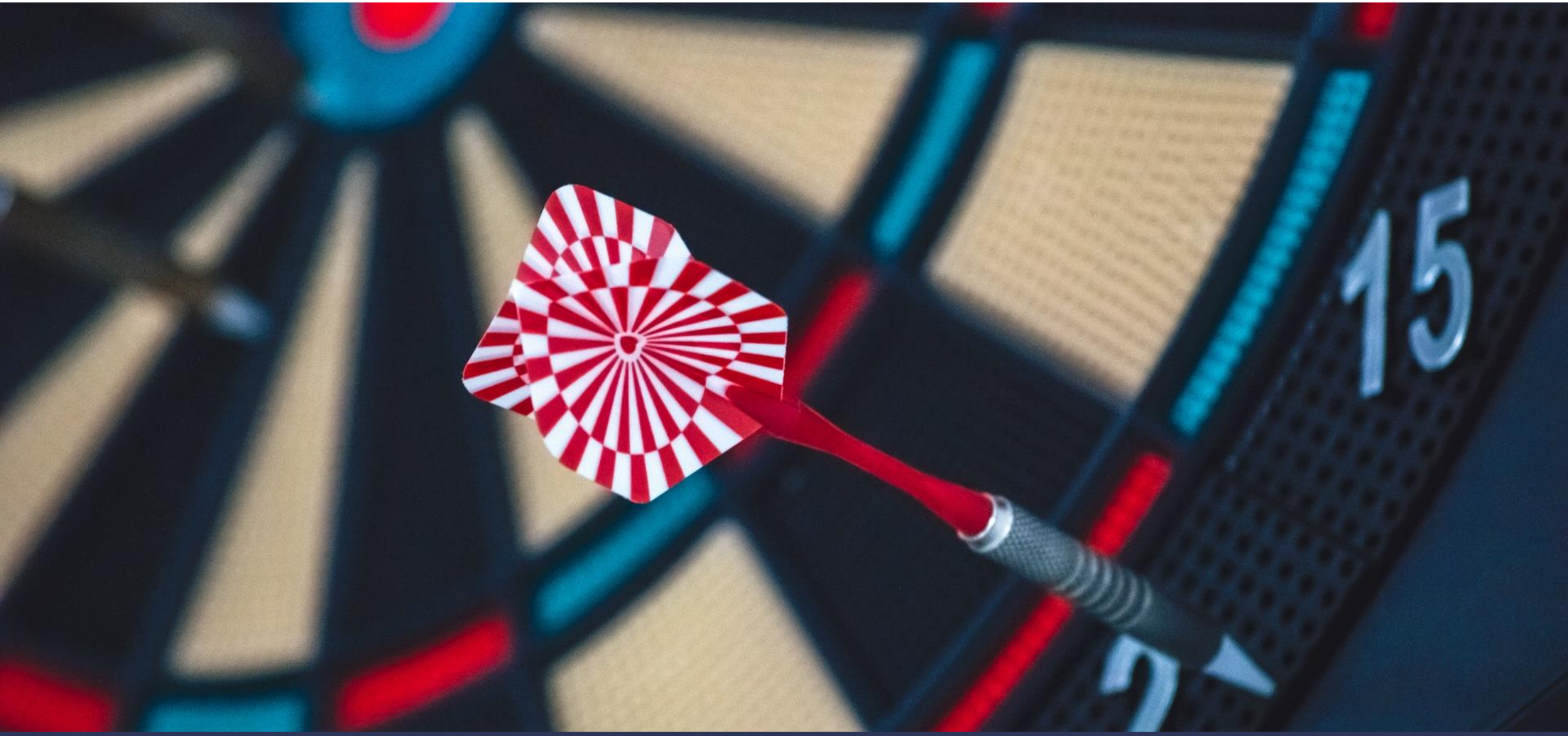
ITDevCon9
European Delphi Conference

Marco Breveglieri

ABLS Team

Actor Model in Delphi

OBIETTIVO



Obiettivo

Creare applicazioni
performanti e scalabili
senza farsi venire il mal di pancia.

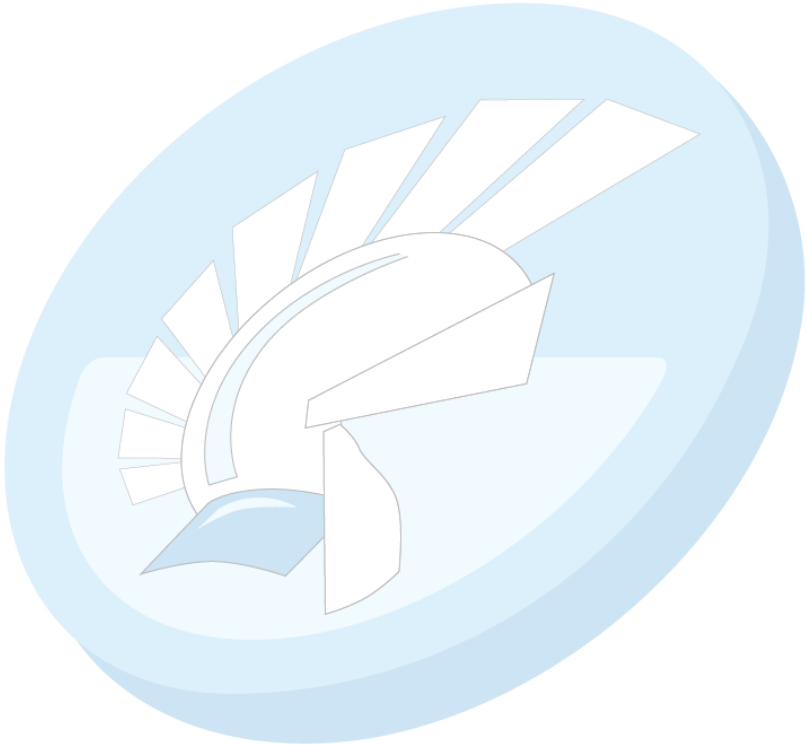
Obiettivo

Quello che vorrei è un programma

- **Semplice**: non è bello perdere anni di vita nel tentativo di coordinare dei thread con una marea di codice
- **Scalabile**: vorrei poter aumentare i dati da elaborare senza un decadimento delle performance a causa di lock
- **Robusto**: non voglio che la mia applicazione abbia comportamenti strani o vada in crash
- **Flessibile**: voglio modificare il codice solo quando si presenta una esigenza di business o una variazione architetturale di alto livello
- **Versatile**: vorrei trasformare il mio progetto in un micro servizio o portarlo su altre piattaforme
- **Responsivo**: voglio che risponda in tempi brevi quando server, senza bloccarsi o consumare CPU a tempo indefinito quando non richiesto

ITDevCon9
European Delphi Conference

PREMESSA



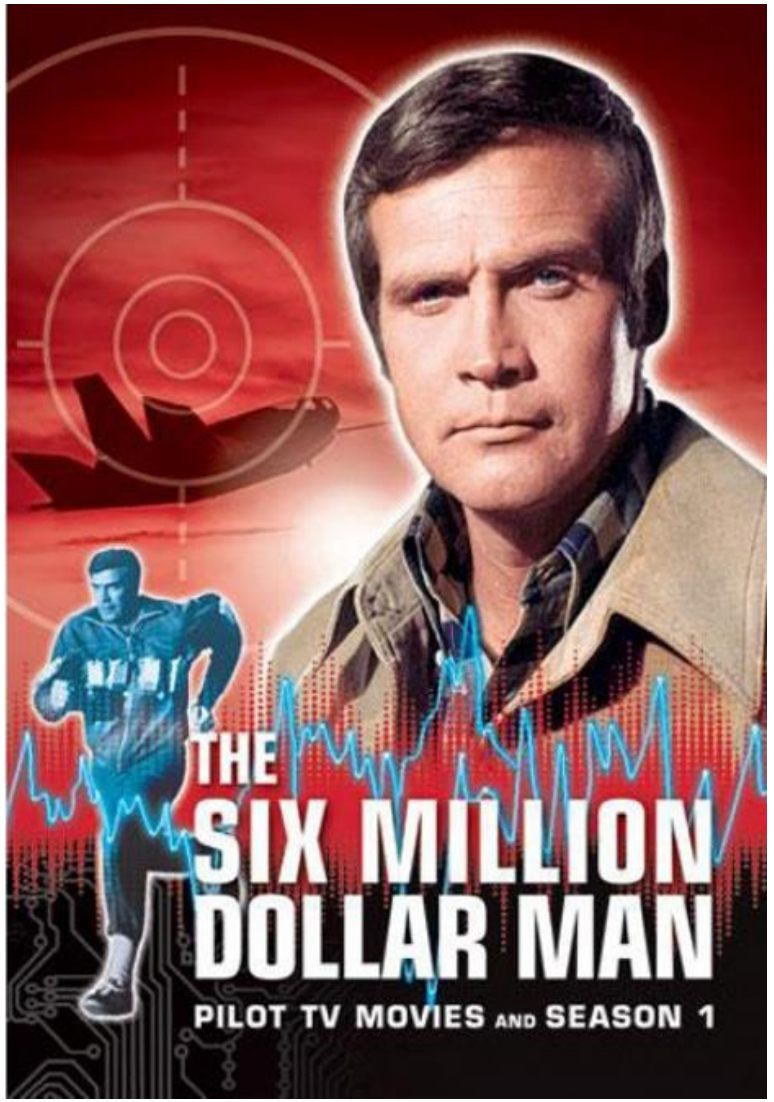
VIVIAMO IN TEMPI DURI...

- Milioni di persone accedono a qualsiasi tipo immaginabile di servizio su Internet
- L'attenzione degli utenti è sempre più ridotta e mirata
  - Sommersi da una valanga di informazioni
  - Qualche istante in più nella risposta può pregiudicare l'acquisto di un prodotto o la lettura di un articolo (*)

(*) Amazon ha determinato che **ogni 100ms** di latenza del sito causa un **-1% delle vendite**
👉https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

Abbiamo le tecnologie!

- Ampia disponibilità di sistemi interconnessi via Internet
  - Server Grid
  - Data Center
  - Infrastrutture Cloud
  - Macchine multi-core
- Costi bassi per le risorse
  - CPU / GPU a 2, 4, 8 o più core
  - RAM ampia e veloce
  - Spazio su dischi (HDD o SSD)
  - Connessione a banda larga



L'EVOLUZIONE DELLA SPECIE

Passato

Un server

Pochi utenti

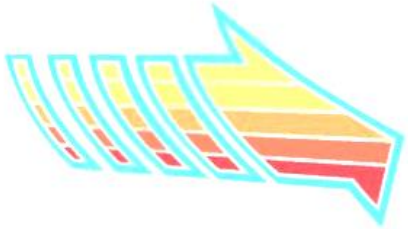
Una CPU

RAM limitata

Spazio su disco limitato

Rete lenta e costosa

Pochi MB di dati



Presente

Più server e sistemi Cloud

Milioni di utenti concorrenti

Multi-core CPU

RAM economica e illimitata

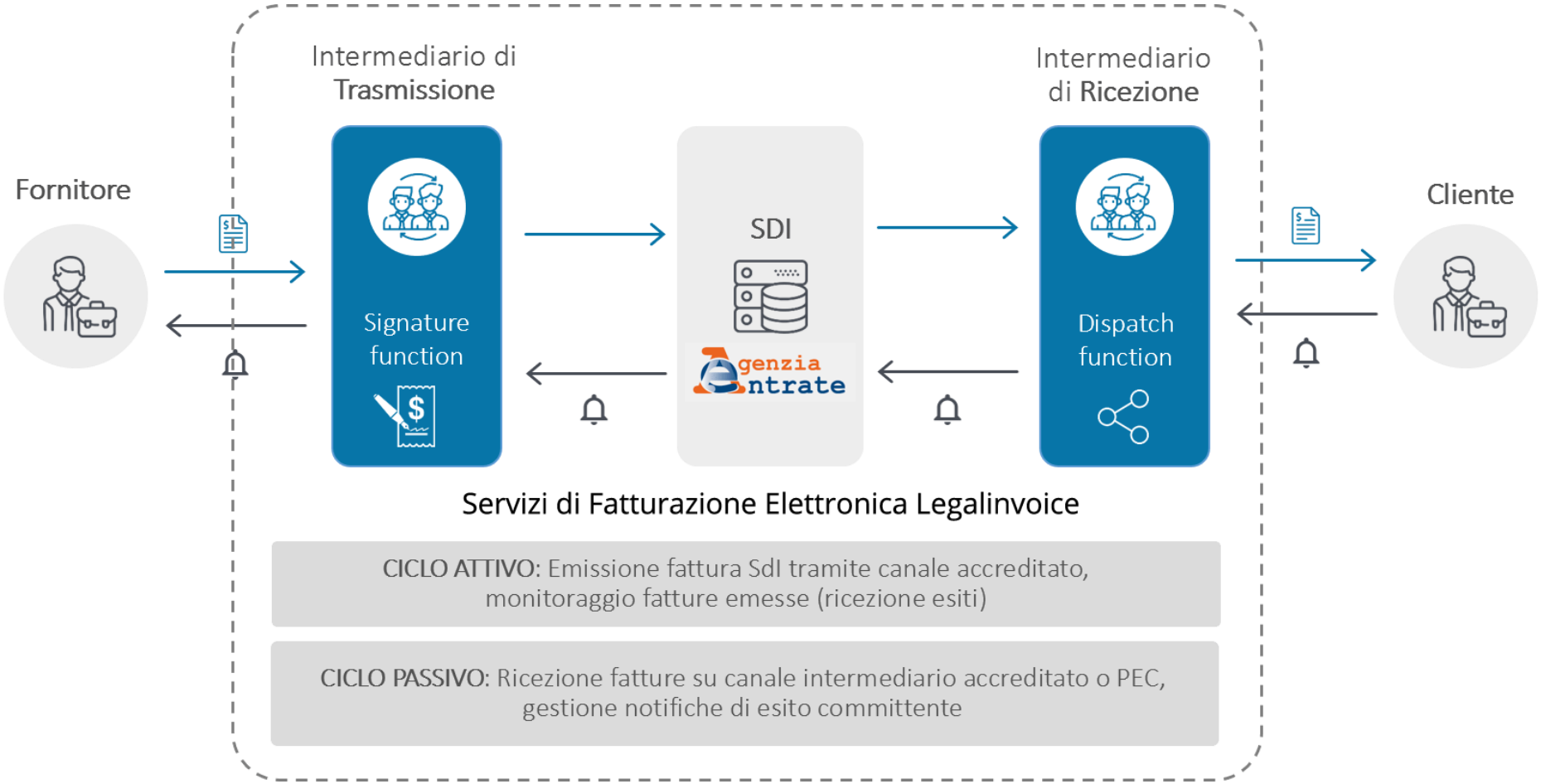
Spazi illimit

Rete ve

TB di dati, fino ai

Back to Basics!

ESEMPIO: ELABORAZIONE FATTURE ELETTRONICHE



Fonte immagine: **www.infocert.it**

ESEMPIO: ELABORAZIONE FATTURE ELETTRONICHE

- Scaricare fatture del ciclo passivo dal Sistema di Interscambio (SdI)
- Alimentare tabelle con i dati e il log degli stati dei flussi elaborati
- Decriptare ed estrarre le fatture dalla "busta" che le contiene
- Riconciliare le fatture con il cliente di appartenenza
- Caricare le fatture all'interno di un portale per la consultazione
- Rendere le fatture ricercabili attraverso campi chiave estratti
- Rendere disponibile il PDF della fattura (generato automaticamente)
- Conservare in archivio una copia del PDF (con fattura XML allegata)

La capacità elaborativa dovrà sostenere il volume sempre più frequente dei documenti in ingresso quando il sistema sarà a regime a inizio 2019.

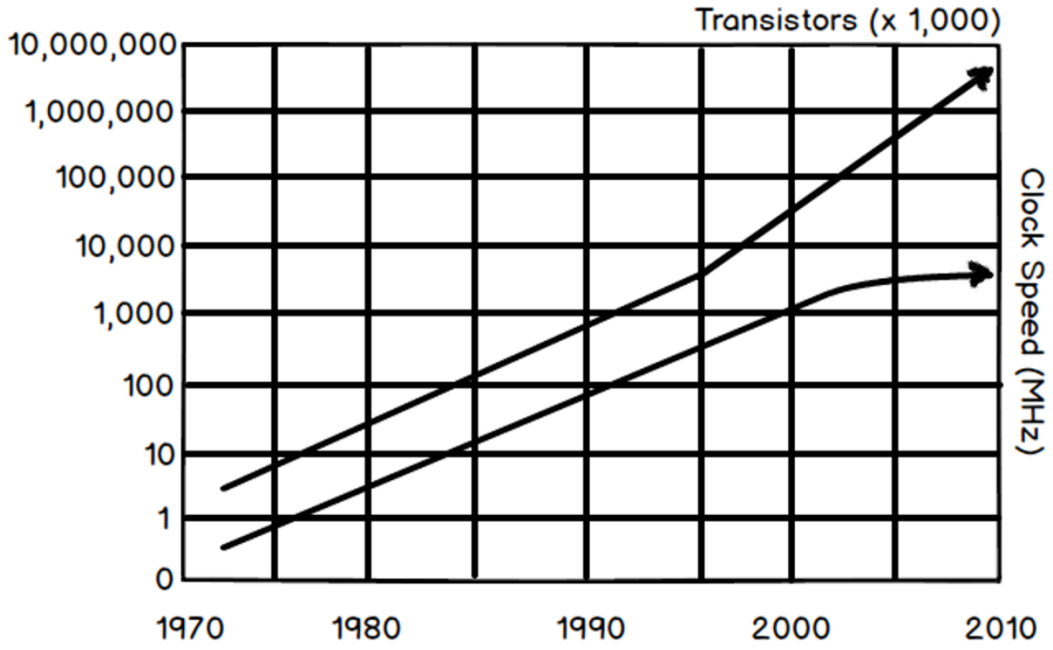
Esame di coscienza

- Perché devo attendere la fine di un file prima di processarne un altro?
- Perché ho un backend multi-core che passa più tempo in "idle" che a lavorare?
- Come posso fare più di una cosa alla volta anticipando le operazioni successive senza tediarmi né impazzire con il debugging?
- Ci sono astrazioni che posso utilizzare?
- Il mio stagista poi saprà capire il codice e manutenerlo senza fare danni?

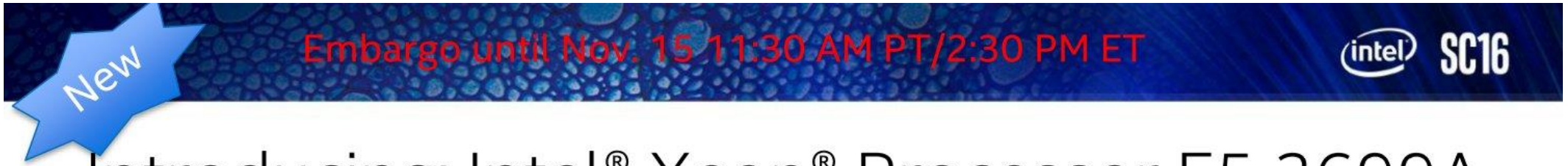
Ci vorrebbe una panacea che permetta di fare un'applicazione con tutti i crismi ma senza richiedere uno sforzo immane.

La sfida

- La velocità di clock dei processori ha smesso di crescere dal 2006
- La legge di Moore si applica ancora, ma solo i core delle CPU sono in aumento
- Oltre un certo grado, "parallelizzare" non è più conveniente (vedi legge di Amdahl)



ANIMA E... *CORE*



Introducing: Intel® Xeon® Processor E5 2699A

Intel® Xeon® Processor E5-2699A v4
(22 cores, 2.40 GHz, 55 MB L3 cache)

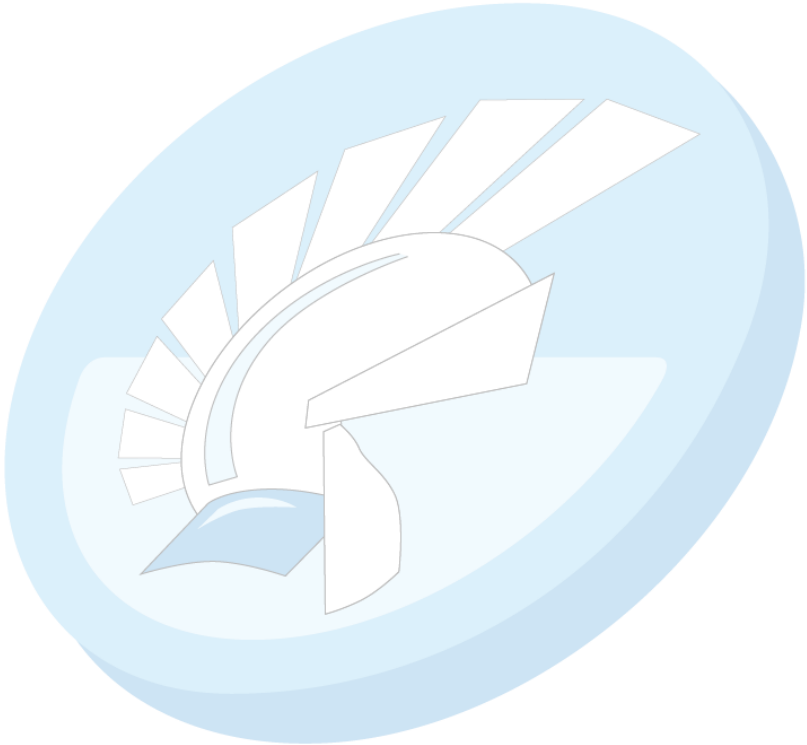
World Record Performance
Up to 4.8% LINPAC gain vs
Intel Xeon processor E5-2699

Results have been estimated or measured based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configurations: Intel internal measurements as of September 2016. Measured see configuration P28. For more information go to http://www.intel.com/performance/datacenter.

COME FARE?

ITDevCon9
European Delphi Conference

MULTITHREADING



Dove si usa?

- Applicazioni di monitoraggio industriale
- Servizi e daemon per elaborazione dati in background
- Supervisioni e applicativi di tipo HMI/SCADA
- Applicazioni moderne e responsive
- Server Web o servizi Internet

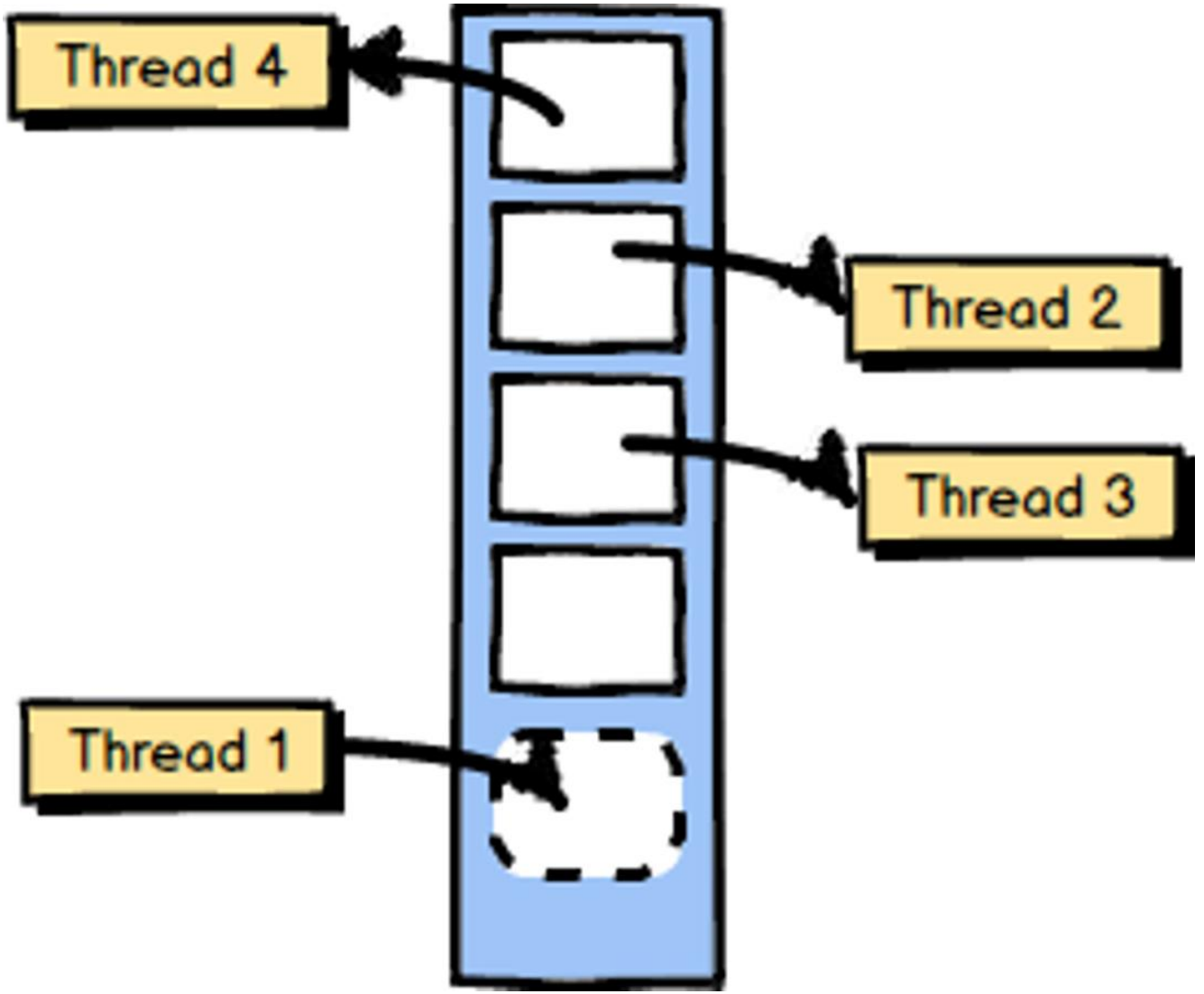
Altre? 🤔

Multithreading in Delphi

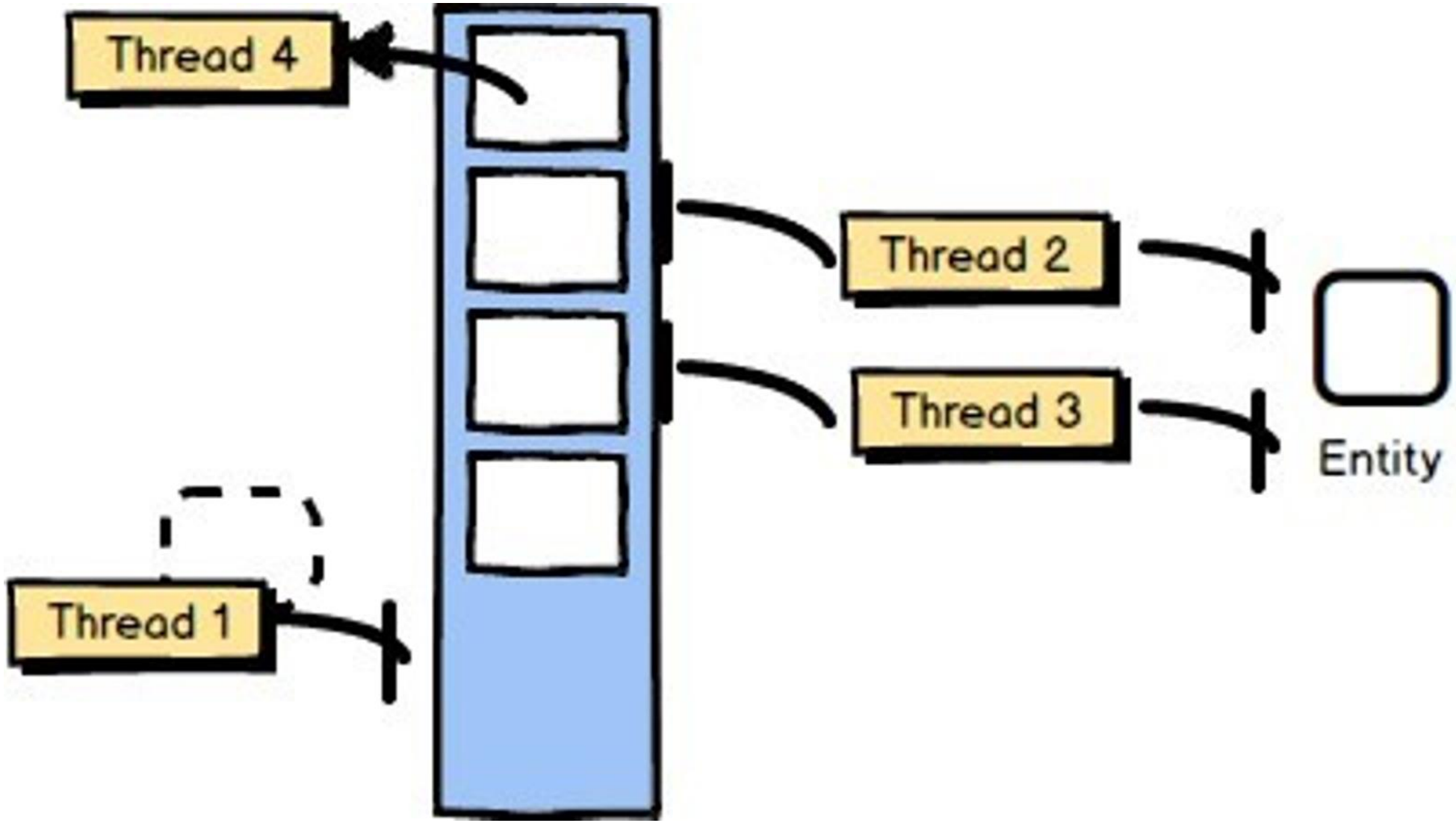
- **`TThread`**
  - Troppo "fisico" (accoppiato alla risorsa che rappresenta)
  - Risorsa comunque disponibile in quantità limitata
  - Privo di supporto a qualsivoglia business logic "out of the box"
  - Difficile gestione automatica del pooling o di thread correlati
- **`TTask`** (*Parallel Programming Library*)
  - Livello di astrazione più alto del thread (vedi *Future*)
  - Adatto a esecuzioni parallele di breve durata
  - Facile abuso del pattern stile *async+await*
  - Progettato per pattern di programmazione asincrona

In generale, sono strumenti che lasciano il controllo di parallelismo/concorrenza nelle mani dello sviluppatore e non forniscono astrazioni per la logica applicativa (business logic). 🤔

THREADING IS EASY!



THREADING IS HARD!



Teoria e pratica

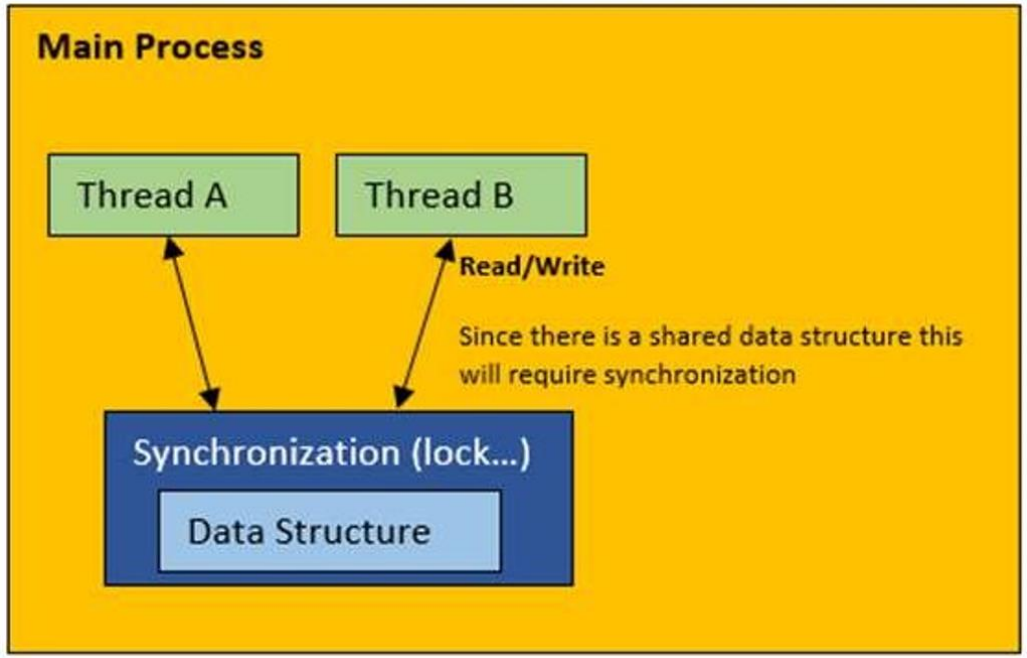
Come ci aspettiamo che funzioni

Teoria e pratica

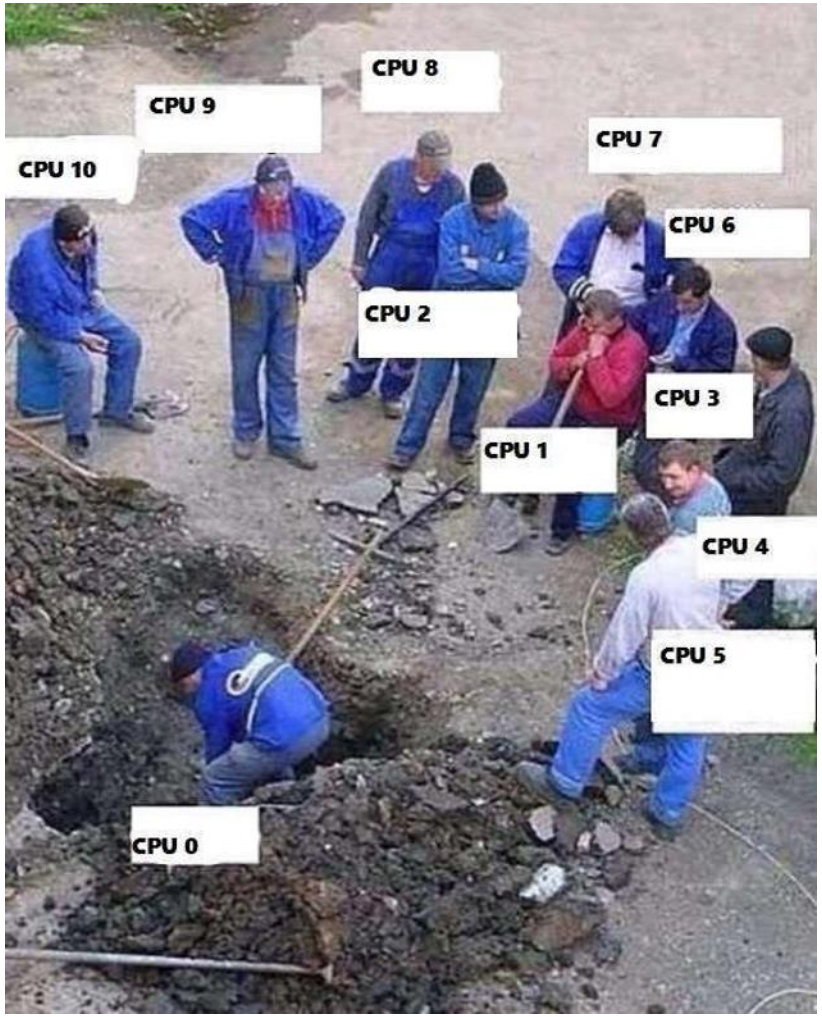
Come funziona in realtà

Shared Data Structure

- Quando due o più thread condividono il medesimo stato
  - L'accesso simultaneo a una risorsa può produrre risultati diversi da quelli attesi
  - La chiamata di metodi (non thread-safe) può portare alla "rottura" dell'oggetto
- E' necessario adottare misure per gestire la race condition (semafori, mutex, critical section, ecc.)
- Tutto questo può portare a
  - Blocking Calls
  - Dead Lock
  - Performance Kill

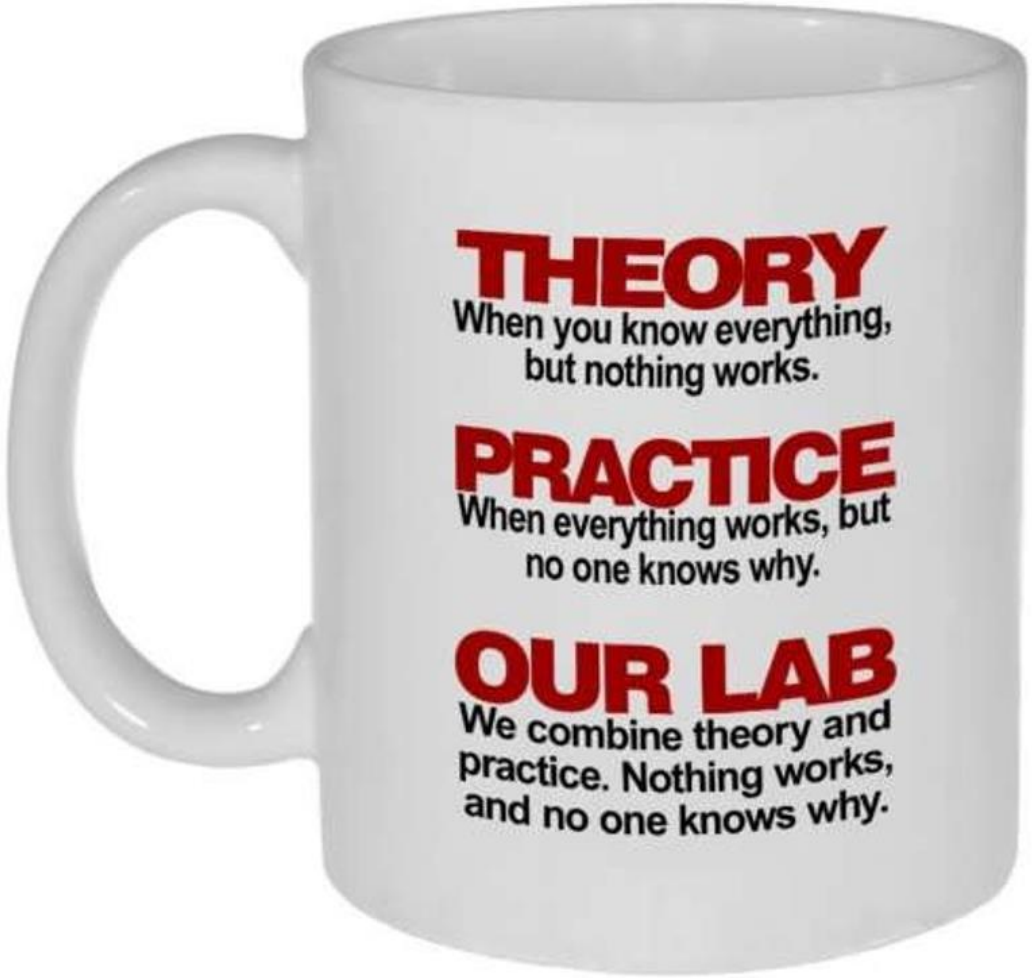


RISULTATO FINALE



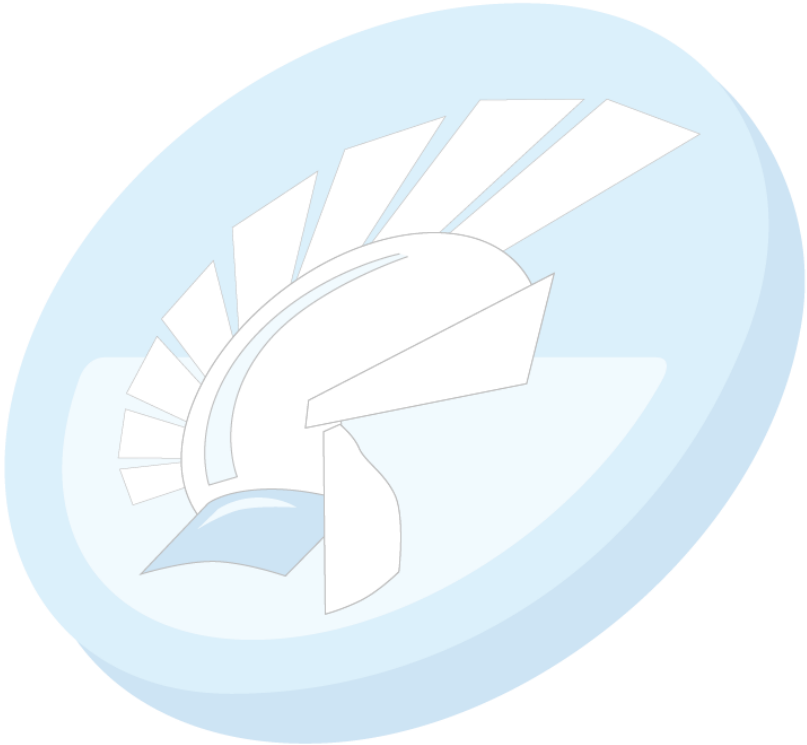
COROLLARI

- Il codice scritto diventa molto complicato e difficile da manutenere
- Il tempo di progettazione delle applicazioni aumenta
- Vi sono troppe potenziali problematiche, anche nascoste
  - Races, dead locks, live locks, lock convoys, cache coherency overheads, lost event notifications, broken serializability, priority inversion, …
- Solo sviluppatori esperti possono comprendere e quindi modificare il codice



ITDevCon9
European Delphi Conference

Actor Model!

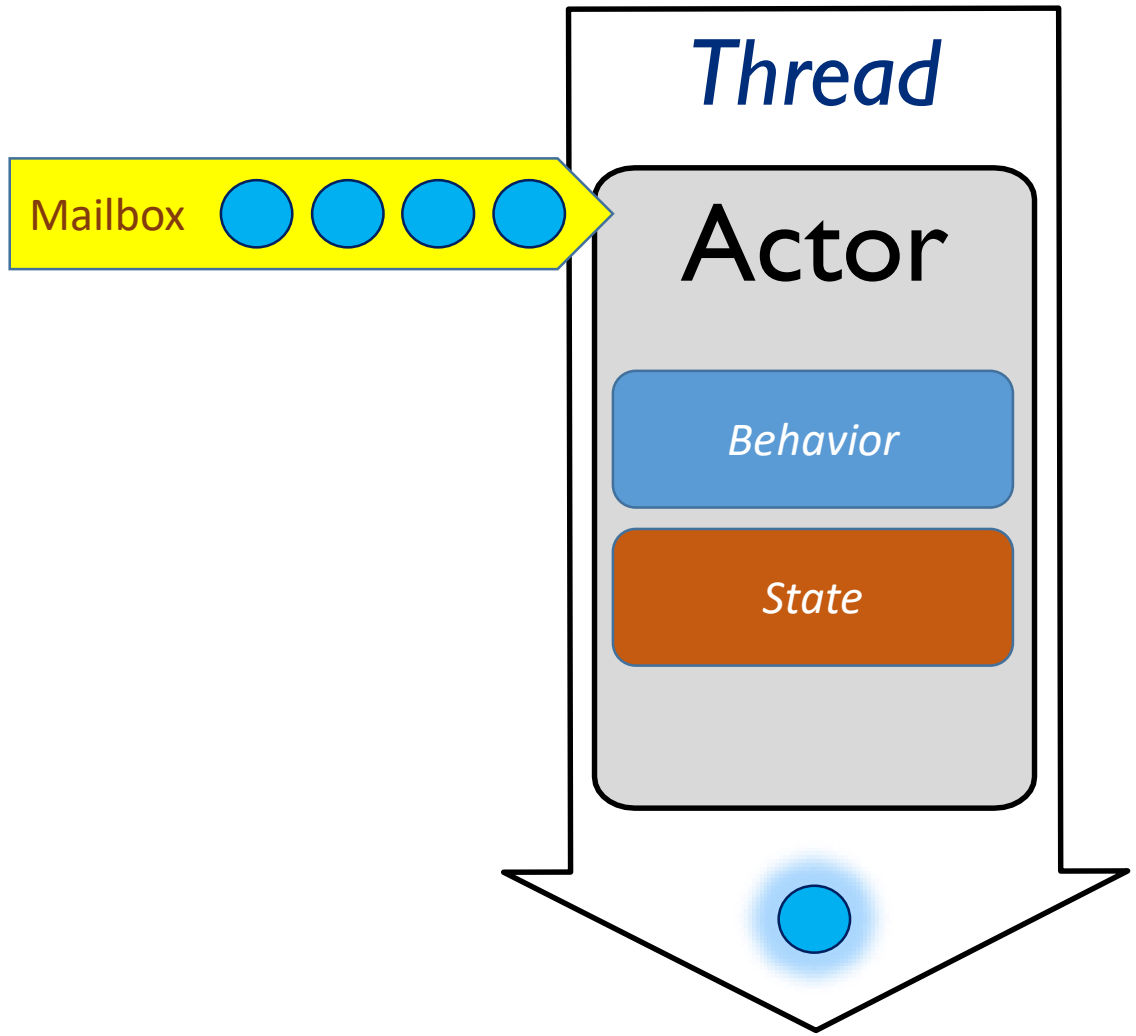


Actor Model

- E' un modello concettuale: definisce regole generali su come i componenti del sistema (attori) debbano interagire tra loro
- Ideato nel 1973 da Carl Hewitt
- Rivisto negli anni '80 da Gul Agha
- Prima implementazione conosciuta da parte di Ericsson
  - Ha costruito un sistema di telecomunicazioni efficiente (ben 99.9999999% di uptime)
    - Distribuito
    - Concorrente
    - Fault-tolerant
  - Utilizzato come modello per Erlang
  - Divenuto open source negli anni '90

ACTOR: IL COMPONENTE FONDAMENTALE

- Oggetti "leggeri" (lightweight)
- Ciascuno esegue nel contesto del proprio thread
- Ciascun attore ha uno stato privato, non condiviso
- Possono scambiare messaggi con altri attori
- I messaggi sono salvati in una mailbox e processati in ordine
- Scalabile e veloce grazie allo stack molto semplificato

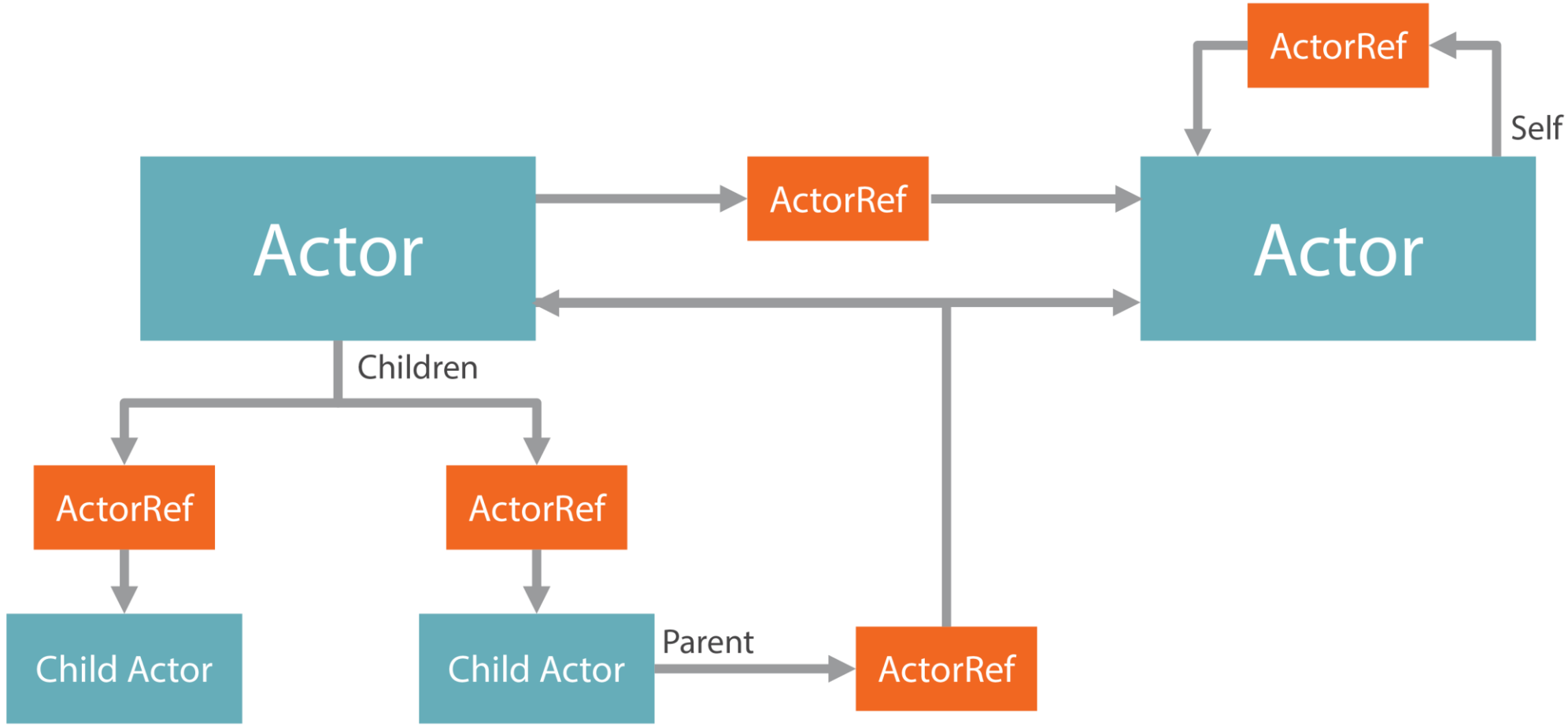


CARATTERISTICHE DELL'ACTOR

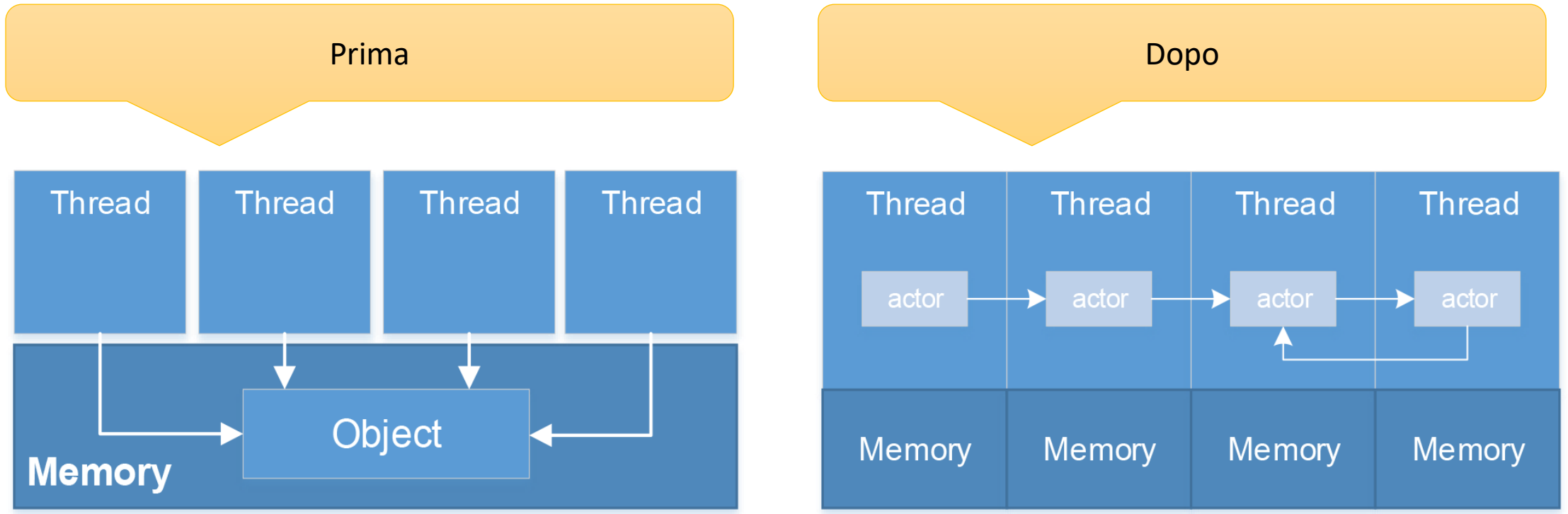
- E' il componente fondamentale dell'architettura
- Esegue compiti piccoli e ben definiti, ossia è specializzato
- Ogni istanza possiede un indirizzo (o *Actor Reference*)
- Può essere eseguito in locale o in un sistema distribuito
- Riceve ed elabora i messaggi della propria mailbox
- Può aggiornare il proprio stato o alterare il proprio comportamento
- Può mandare messaggi ad altri attori in modo asincrono
- Può creare altri attori (figli) per delegare compiti più specifici

IMPORTANTE: ogni attore **processa sempre un solo messaggio per volta**!

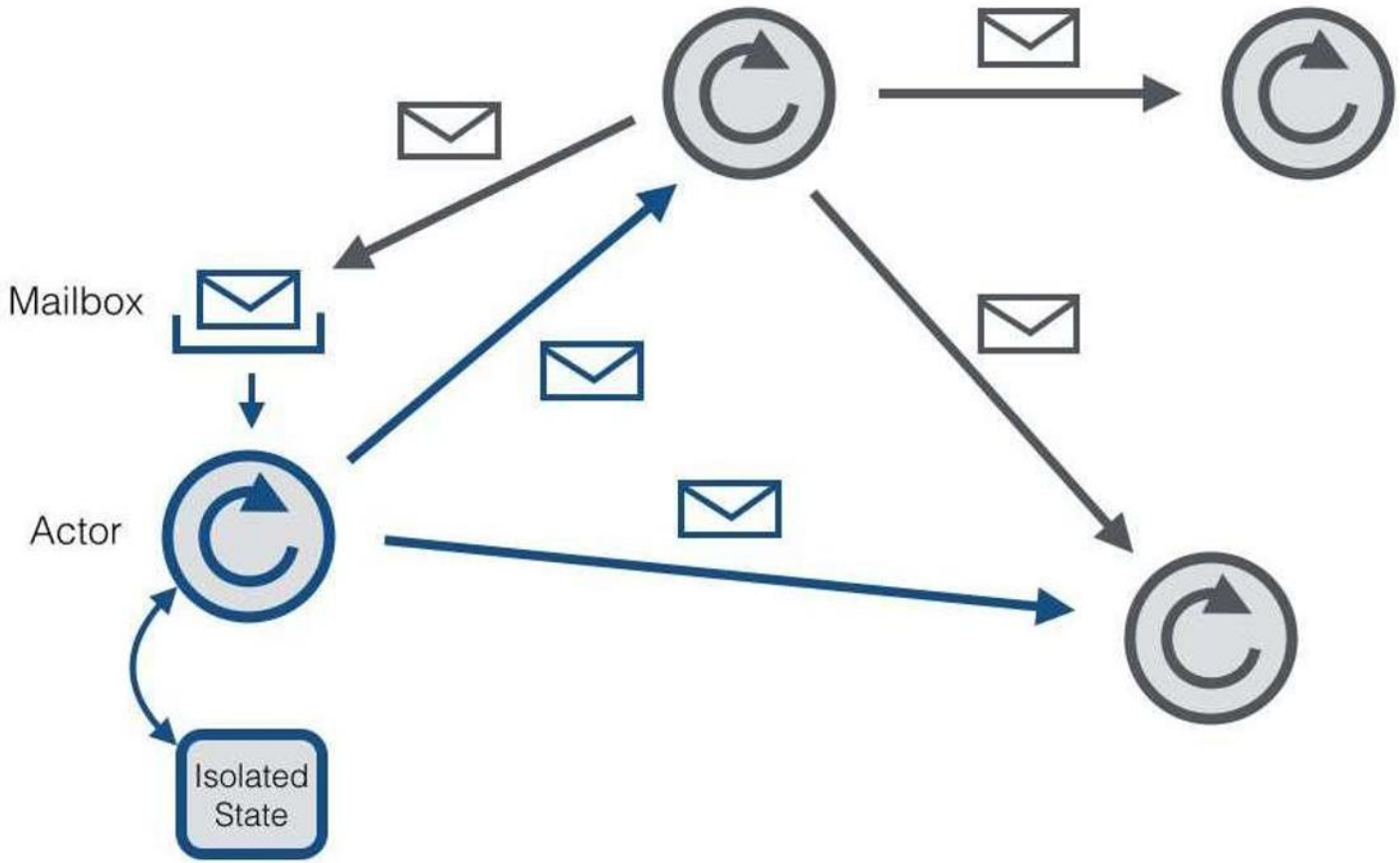
Uso di Actor Reference



Memory Model

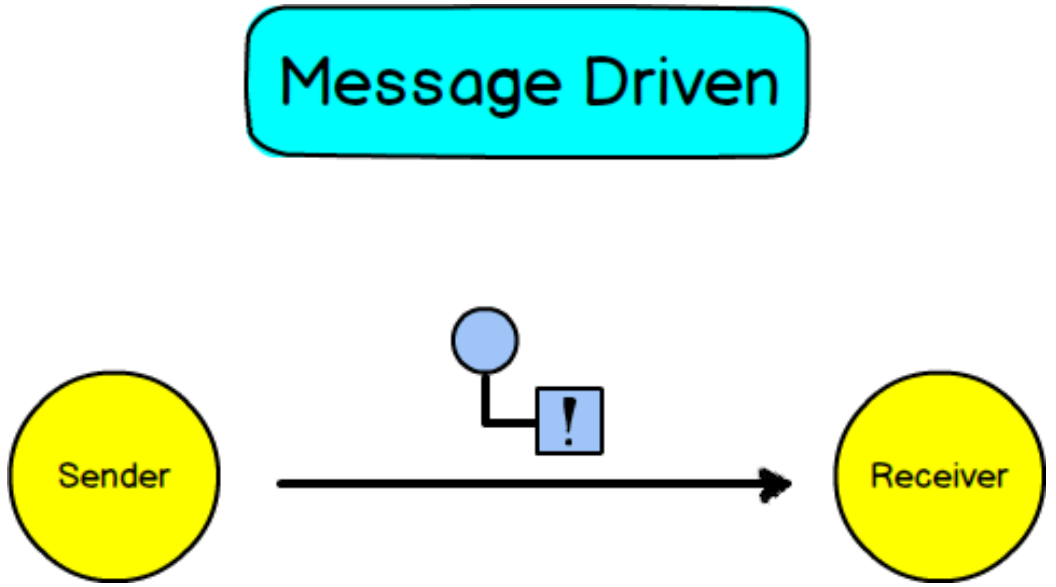


SCAMBIO DI MESSAGGI



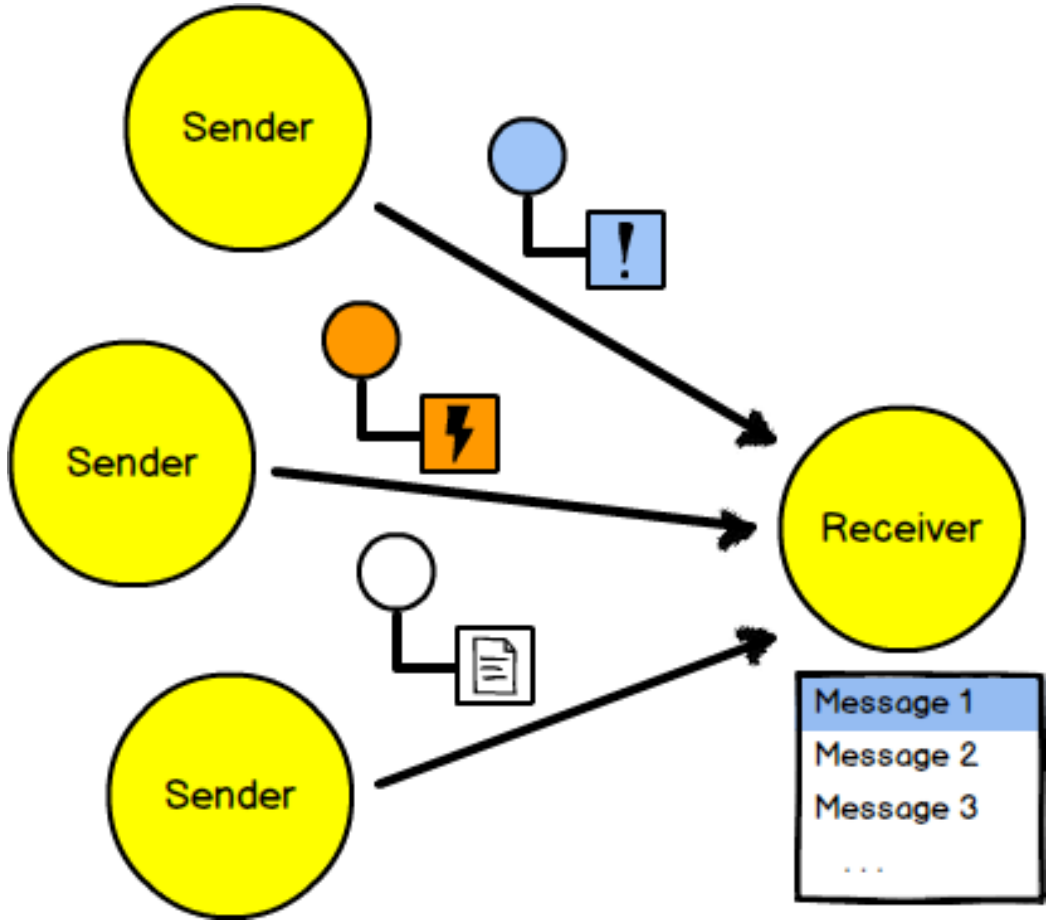
SCENARI

Direct Asynchronous Messaging



SCENARI

Lock Free, Share Nothing

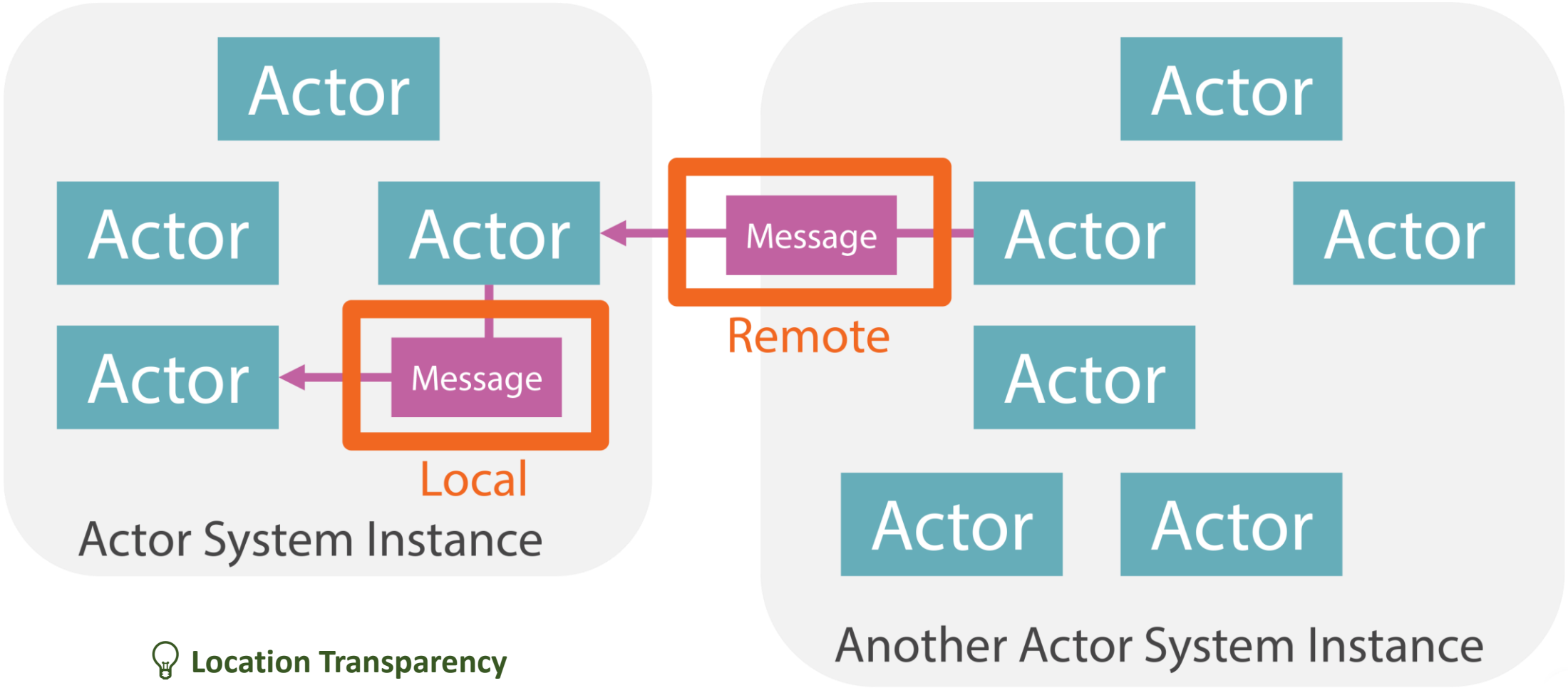


Messaggio

- Viene veicolato tramite un **PODO** (*Plain Old Delphi Object*), ossia l'istanza di una semplice classe dedicata
- Le istanze dei messaggi "dovrebbero" essere immutabili
  - Gli attori non dovrebbero alterare lo stato degli oggetti dei messaggi
  - Il messaggio dovrebbe contenere una copia dei valori che appartengono allo stato privato di un altro attore
- Gli attori possono cambiare stato/behavior rispondendo ai messaggi
- Il framework può supportare messaggi speciali (es. *PoisonPill* )

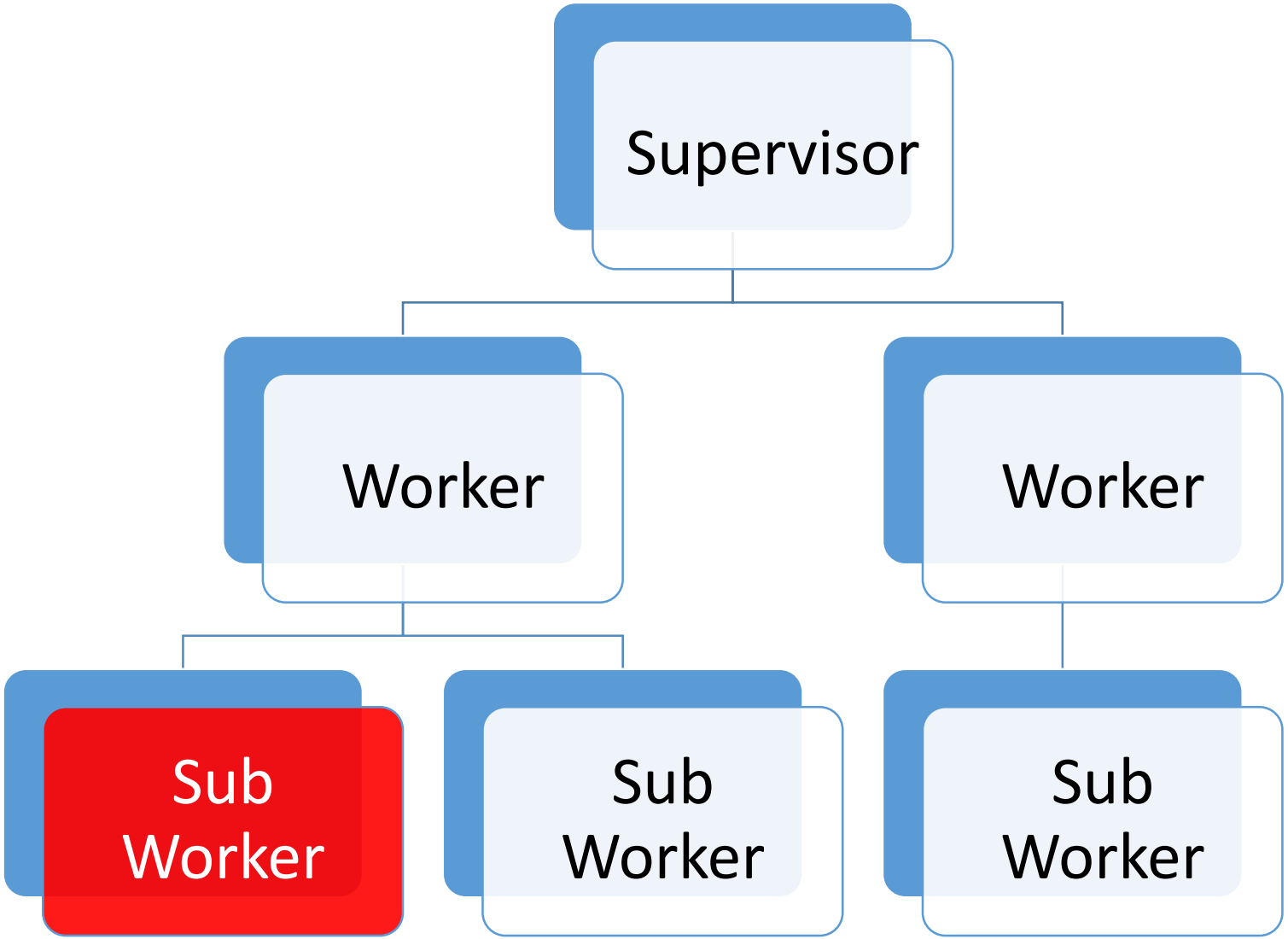
In generale, occorre fare attenzione a **non modificare o alterare lo stato di un oggetto condiviso**, altrimenti si ricade nelle problematiche tradizionali del multithreading.

Actor System

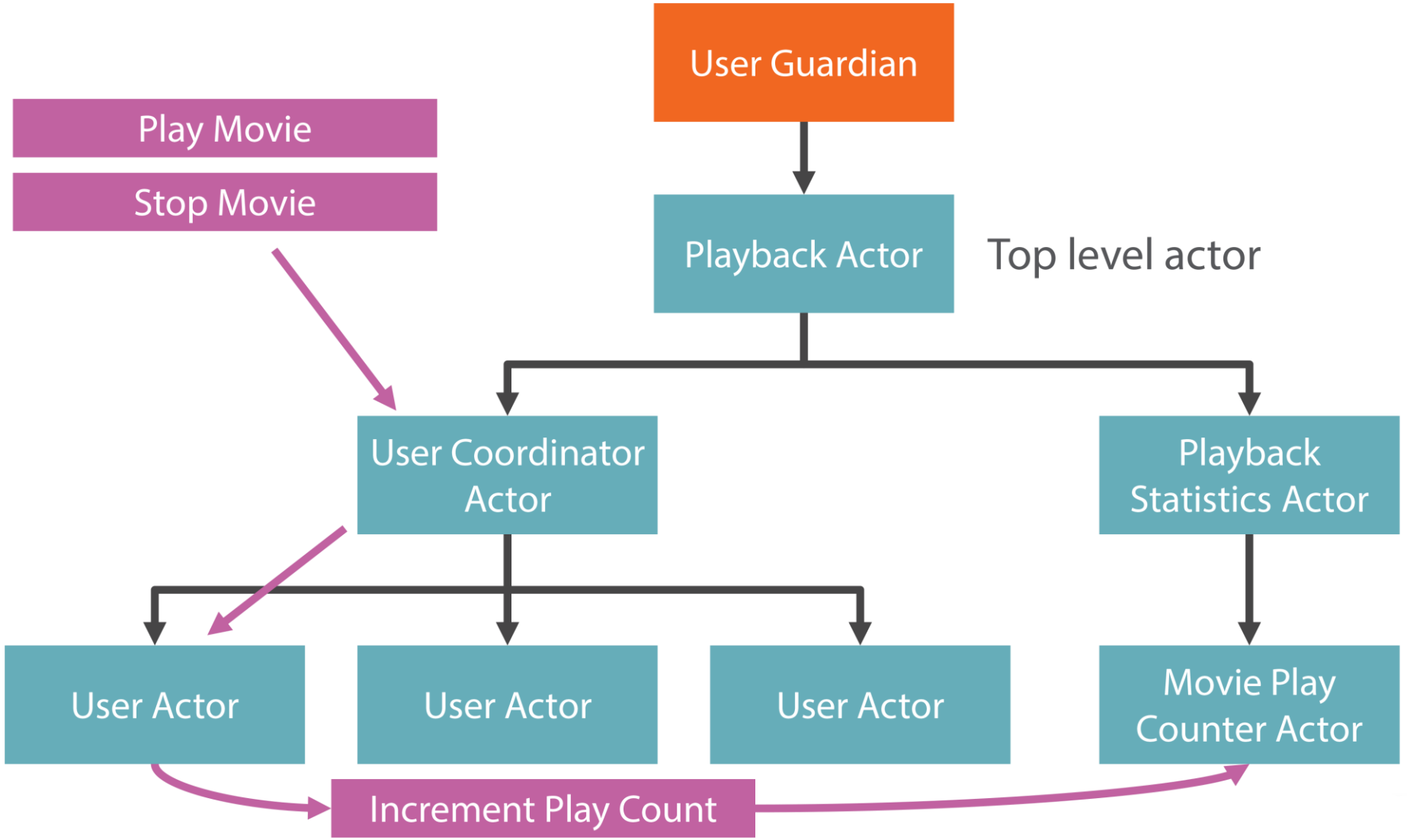


Location Transparency

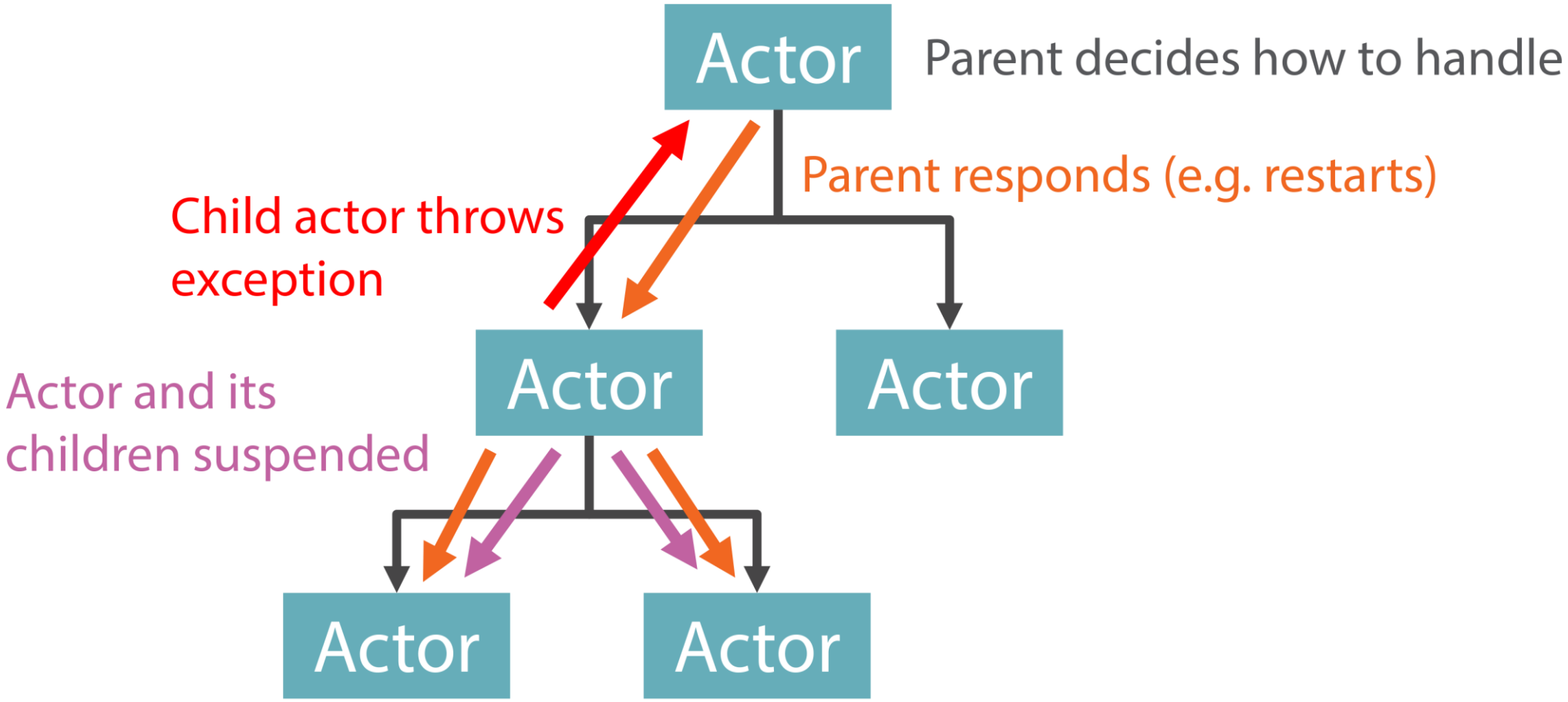
Fault Tolerance



Child Actors /1

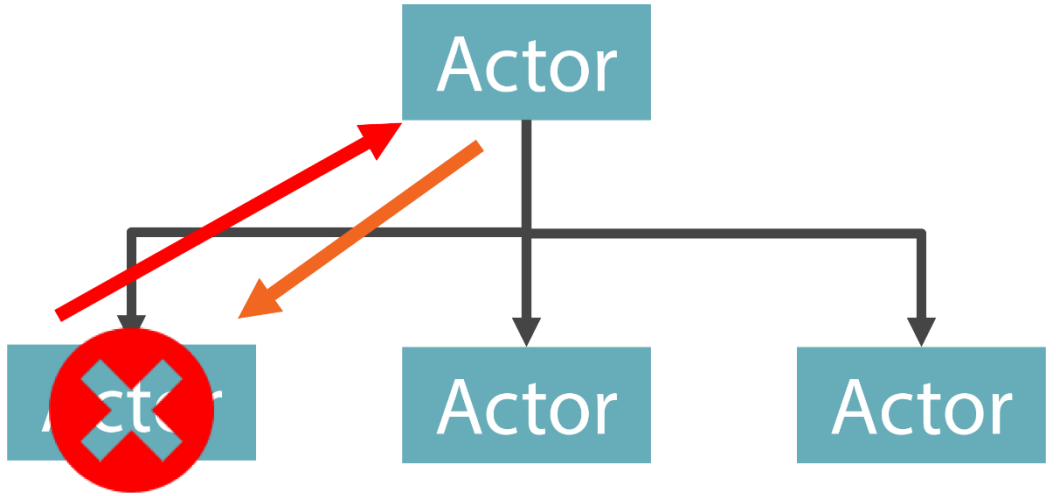


Child Actors /2



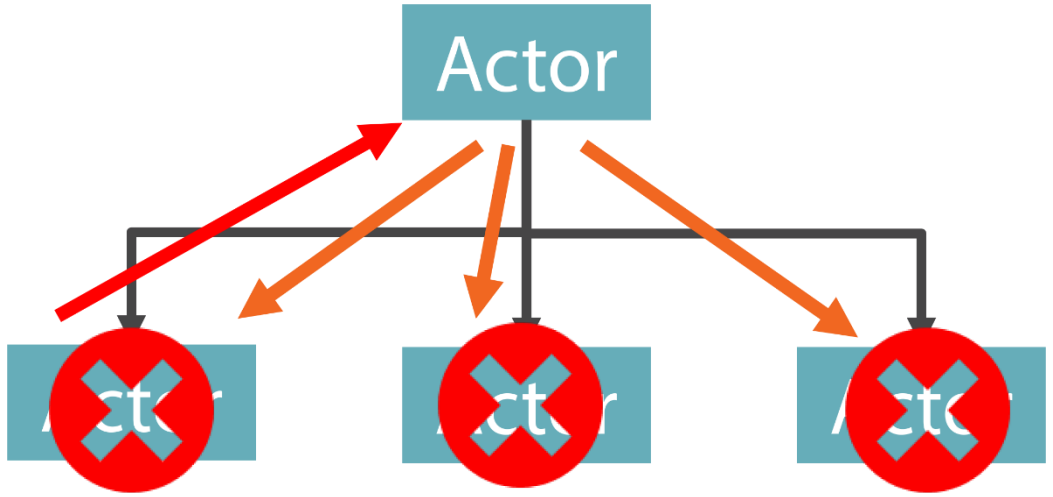
Supervision Strategy /1

One For One



Supervision Strategy /1

One For All



Obiettivi

L'architettura consente di

- Evitare che i thread collidano sui dati
- Utilizzare dei task comunque leggeri e specializzati
- Scambiare messaggi in modo asincrono (no attese)

e semplifica la realizzazione di sistemi

- Scalabili
- Concorrenti
- Elevate prestazioni
- Bassa latenza

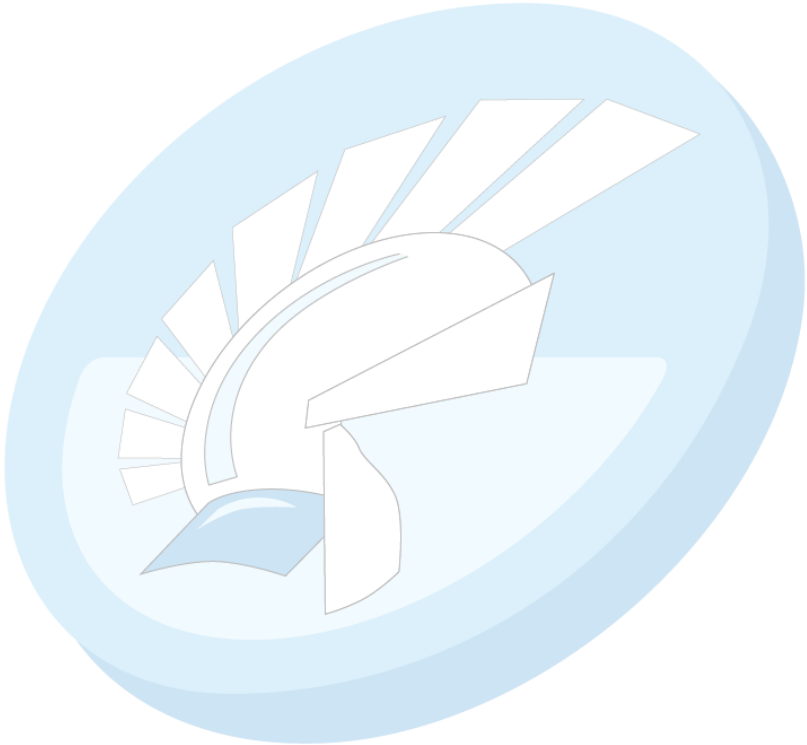
Possibili applicazioni

- Applicazioni transazionali (finanziarie, social media, telecomunicazioni)
- Batch (tramite divisione del carico di lavoro tra più attori)
- Microservice (*REST, GraphQL, System Integration*)
- Comunicazioni (*Chat, Real Time Notifications*)
- Gioco (*Gestione multiplayer*)
- Elaborazione numerica (*Business Intelligence, Data Mining*)
- IoT (*Ingestion del flusso di dati*)

e molte altre…

ITDevCon9
European Delphi Conference

DALLA TEORIA ALLA PRATICA



Implementazioni

- Linguaggi di programmazione
  - Erlang
  - Elixir
  - Pony
  - Scala
- Librerie e framework
  - Akka (*Java*), Akka.NET (*.NET Framework/Core*)
  - Proto.Actor (*Go, .NET*)
  - **Ikaria** (*Delphi*)
- Architetture
  - Microsoft Orleans
  - Service Fabric

DEMO!

"Talk is cheap, show Me the code"

- Linus Torvalds

ALTERNATIVE

- Sviluppo in Delphi
  - *System.Messaging* (sistema di messaging integrato)
    👉 http://docwiki.embarcadero.com/CodeExamples/Tokyo/en/System.Messaging_(Delphi)
  - *Delphi Event Bus* (by Daniele Spinetti)
    👉 https://github.com/spinettaro/delphi-event-bus
- Tools
  - *Oxygene* (*Delphi .NET*) per Akka
    👉 https://www.elementscompiler.com/elements/oxygene/
- Architetture
  - *RabbitMQ*
    👉 https://www.rabbitmq.com/

DEB
DELPHI EVENT BUS

RabbitMQ

Mi piacerebbe crearne uno! 😳

About me

Marco Breveglieri

Software Developer, Trainer and Consultant

@ABLS Team (Reggio Emilia)

- Homepage: www.breveglieri.it
- Blog: www.compilaquindiva.com
- Podcast: www.delphipodcast.com

Proud member of: DELPHI CLUB ITALIA

Q & A


ONE DOES NOT SIMPLY
GIVE A PRESENTATION WITHOUT Q&A
imgflip.com

GRAZIE!