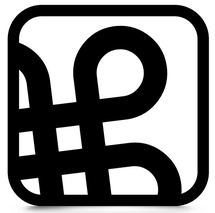


Macoun

ProjectWizards

Threads sind böse.

Frank Illenberger

Ablauf

- Kurze Einführung
- Sensibilisieren für Probleme von Nebenläufigkeit
- Beispielhafte API-Entwürfe mit Grand Central Dispatch

Einführung

Threads

- Sind Teil eines Prozesses
- dienen zum parallelen Erledigen mehrerer Aufgaben
- Prozesse starten mit dem Main-Thread und können weitere Threads erzeugen.
- haben gleichen virtuellen Speicherraum und Rechte
- Eigener Stack und eigene Registersätze

Threads

- Kernel-Scheduler steuert Ablauf
  - quasi-parallel (präemptiv) auf einem Kern
  - oder wirklich parallel auf mehreren Kernen
  - Keinerlei Kontrolle für den Entwickler

Wozu Nebenläufigkeit?

- Besseres Ausnutzen der Prozessorkerne
- Verhindern, dass Main Thread blockiert
- Asynchrone Abläufe (Netzwerk)
- Hintergrundaufgaben (Berechnungen)

Nebenläufigkeit

- Einfache APIs wie NSThread oder NSOperationQueue
- Aber:
  - Korrekte Implementierungen sind subtil schwierig!

Generelle Probleme

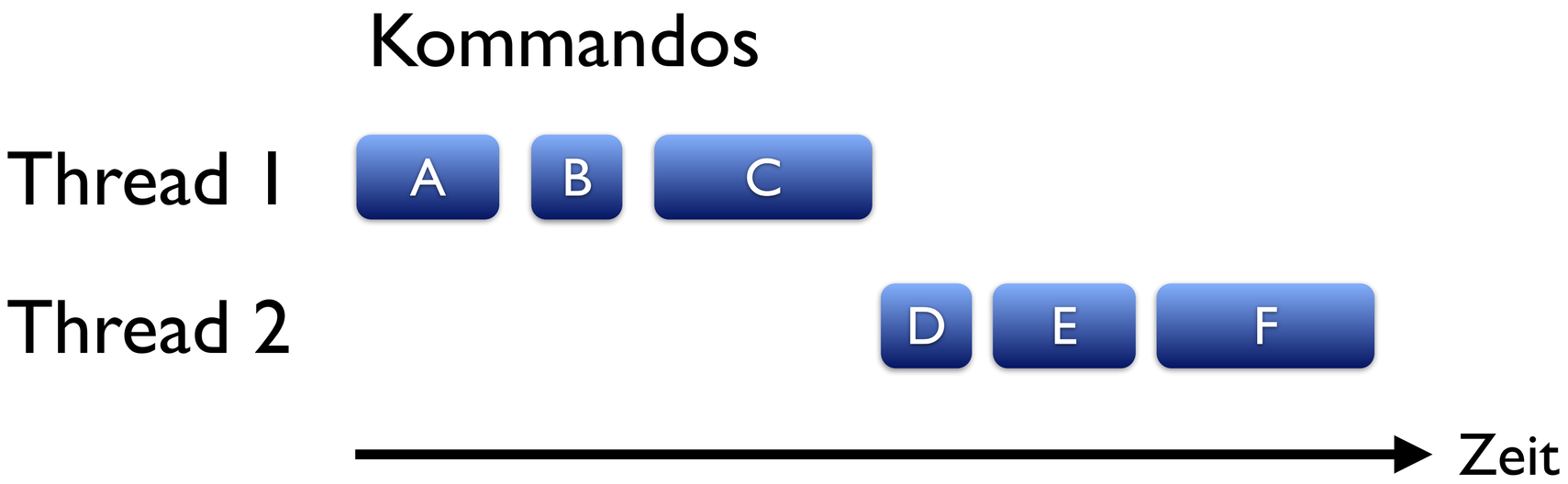
- Race Conditions
- Deadlocks

Race Conditions

- Durch Threading wird der Programmablauf grundlegend indeterministisch
  - Kombinatorische Explosion der Verschachtelungsmöglichkeiten

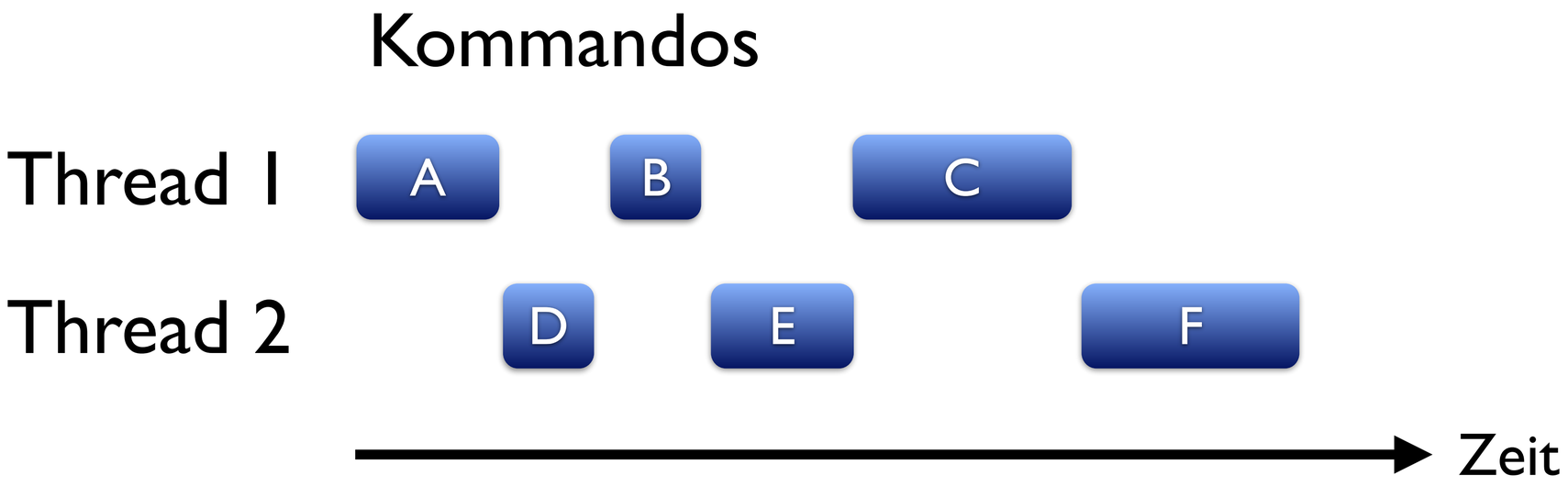
Verschachtelungen

Präemptiv mit einem Kern



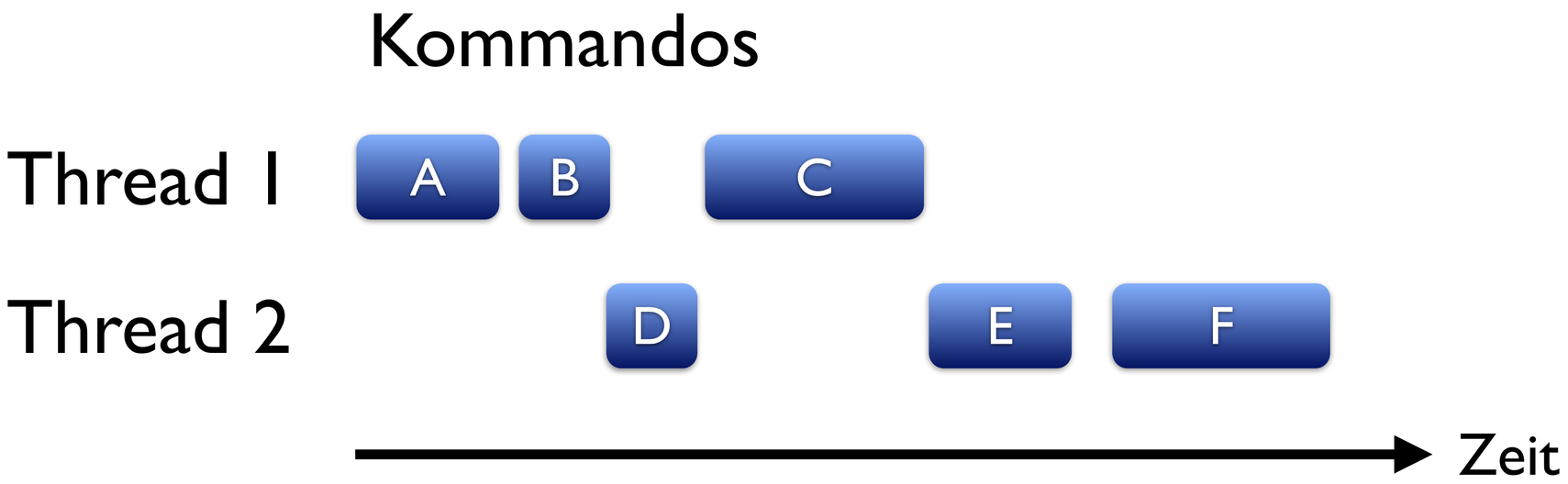
Verschachtelungen

Präemptiv mit einem Kern



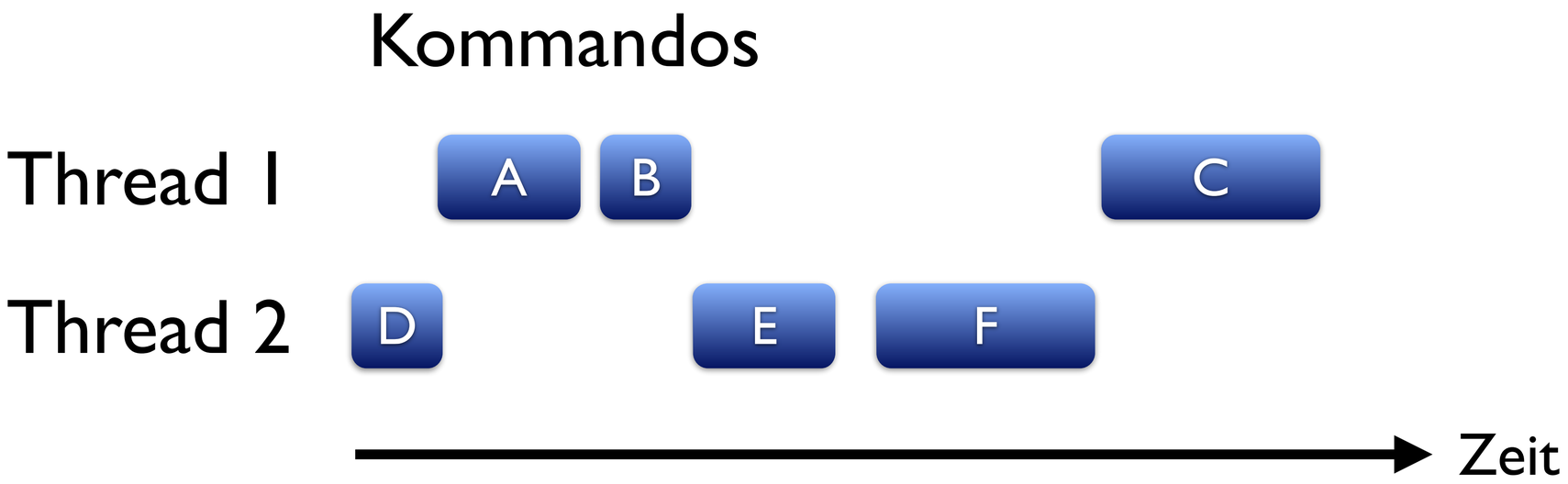
Verschachtelungen

Präemptiv mit einem Kern



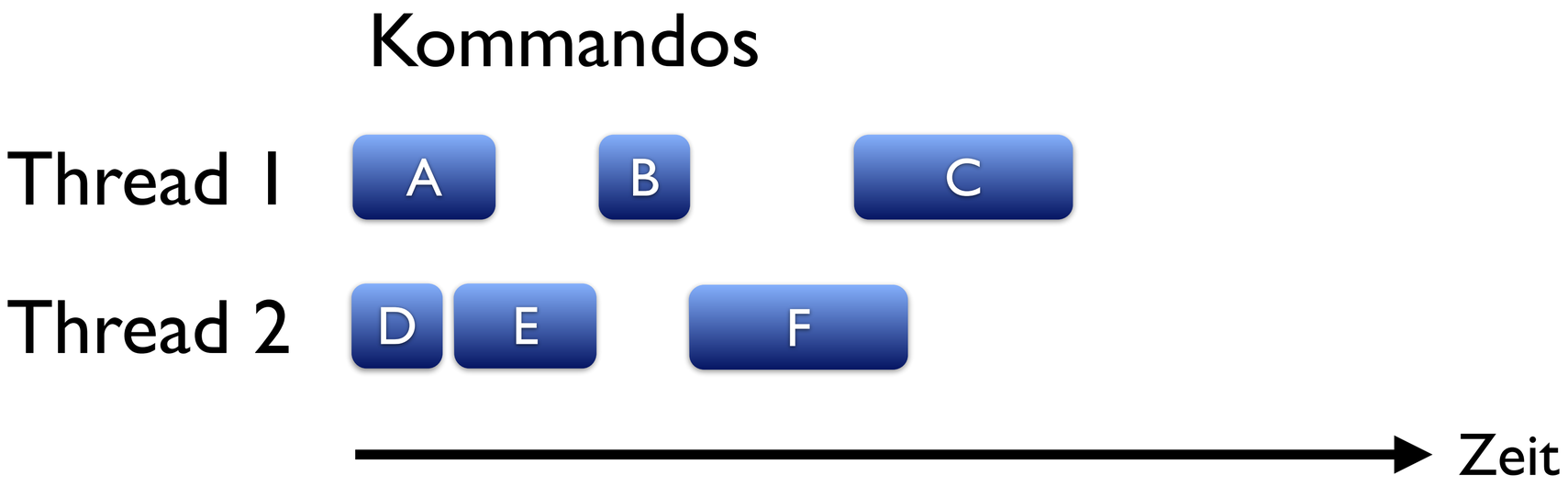
Verschachtelungen

Präemptiv mit einem Kern



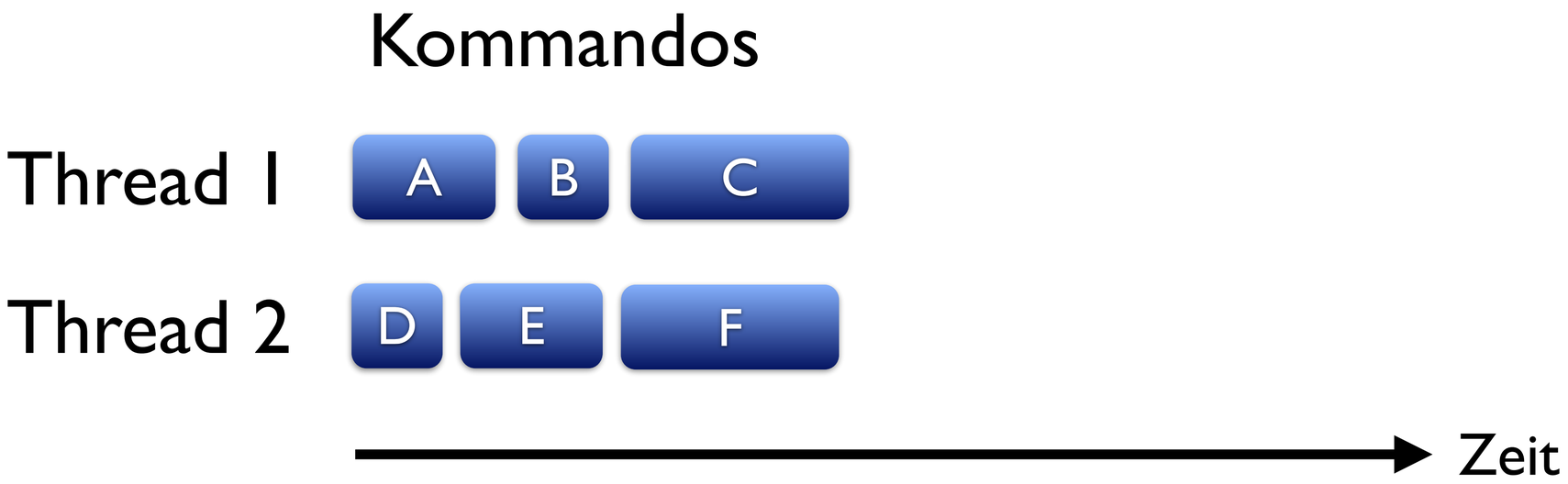
Verschachtelungen

Mit mehreren Kernen



Verschachtelungen

Mit mehreren Kernen



Race Conditions

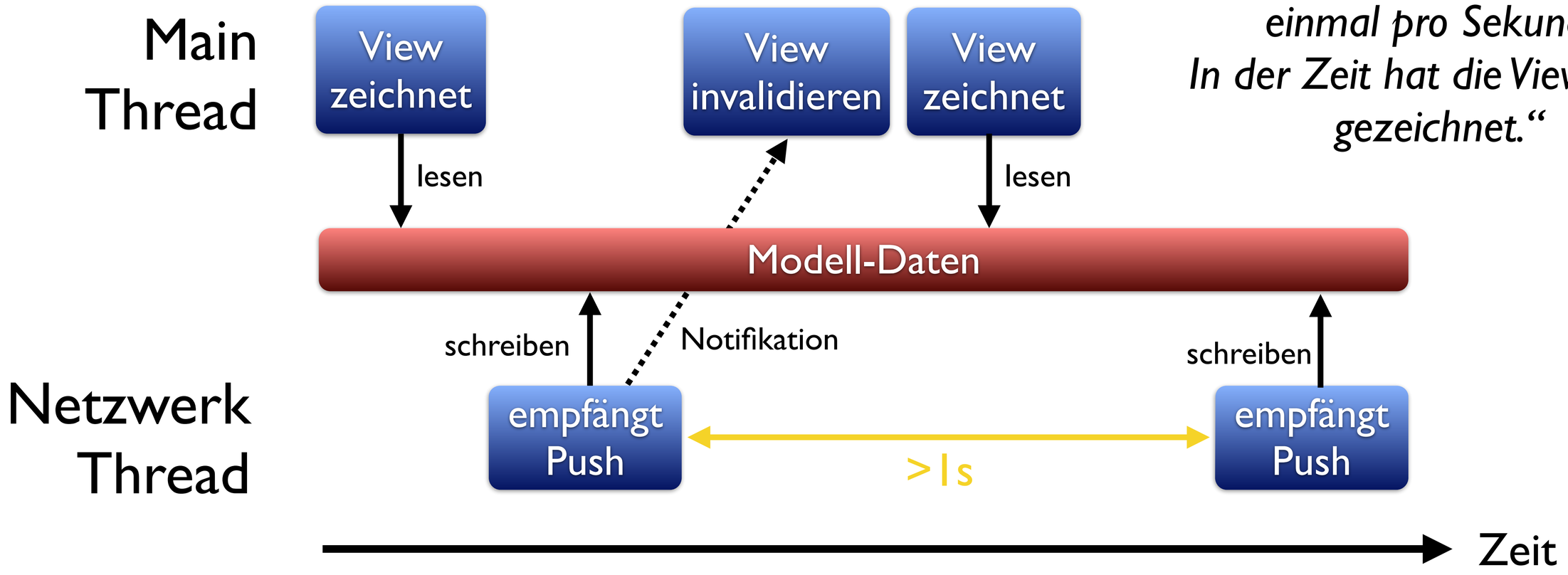
- Durch Threading wird der Programmablauf grundlegend indeterministisch.
  - Kombinatorische Explosion der Verschachtelungsmöglichkeiten
  - Wenn mehrere Threads dabei die gleichen Daten verwenden, können ungewollte Zustände entstehen.

Race Conditions

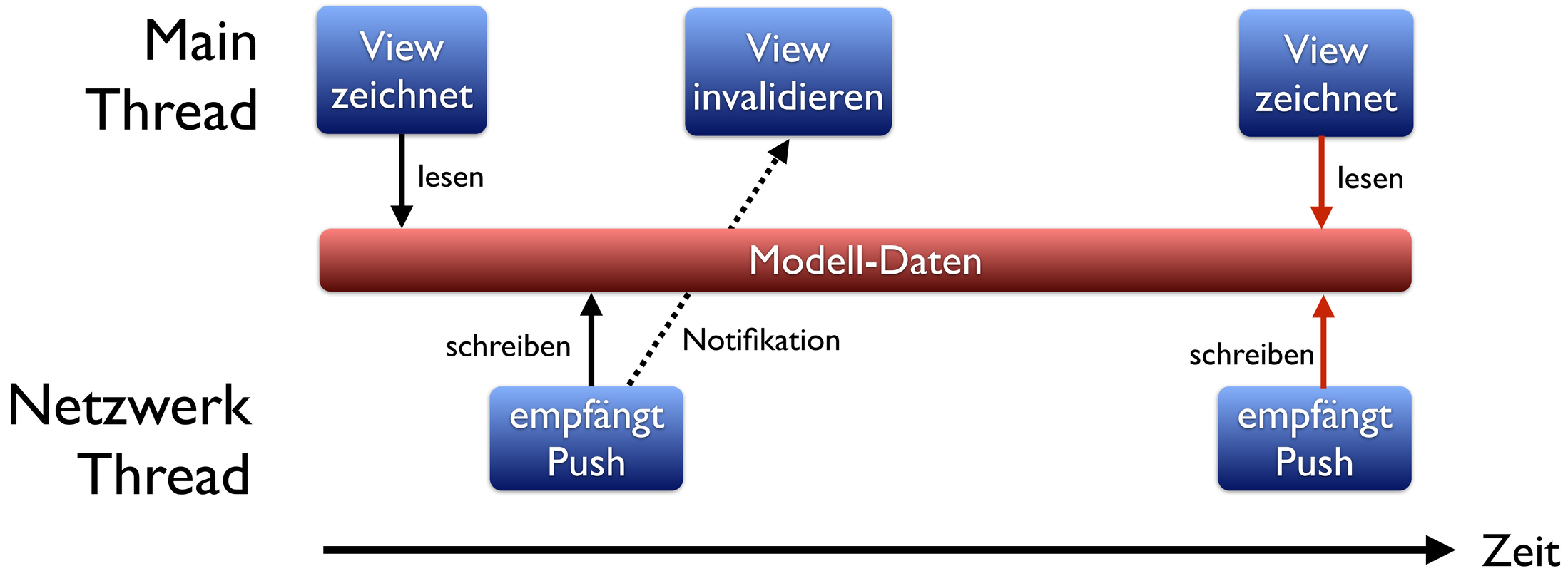
- Fehler treten nur sporadisch oder nur auf gewissen Maschinen auf
- Die eigentlichen Fehlerursachen sind extrem schwer zu debuggen
- Schwer automatisiert zu testen
- Sehe ich häufig in Code Reviews!
  - Sind harte Fehler!

Anti-Beispiel

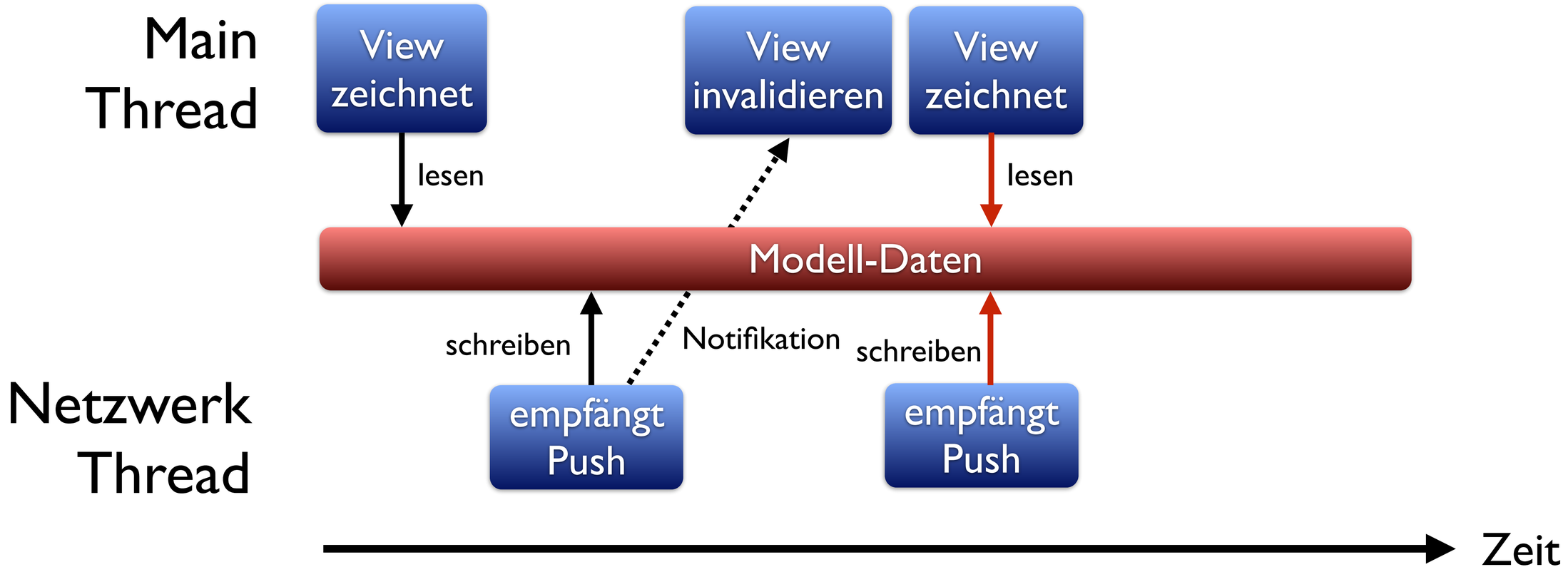
„Der Server sendet maximal einmal pro Sekunde. In der Zeit hat die View fertig gezeichnet.“



Anti-Beispiel



Anti-Beispiel



Demo

Race Conditions

- Gegenmaßnahmen:
  - Keine gemeinsamen Daten
    - Ist für wenige Probleme geeignet
  - Immutable Objects
    - Wenn Daten nicht verändert werden, stört Zugriff von mehreren Threads nicht

Lazy Getter

```
- (id)expensiveValue
{
   if(!_expensiveValue)
      _expensiveValue = [self calculateExpensiveValue];
   return _expensiveValue;
}
```

Objekte mit Lazy-Getter sind nicht immutable!

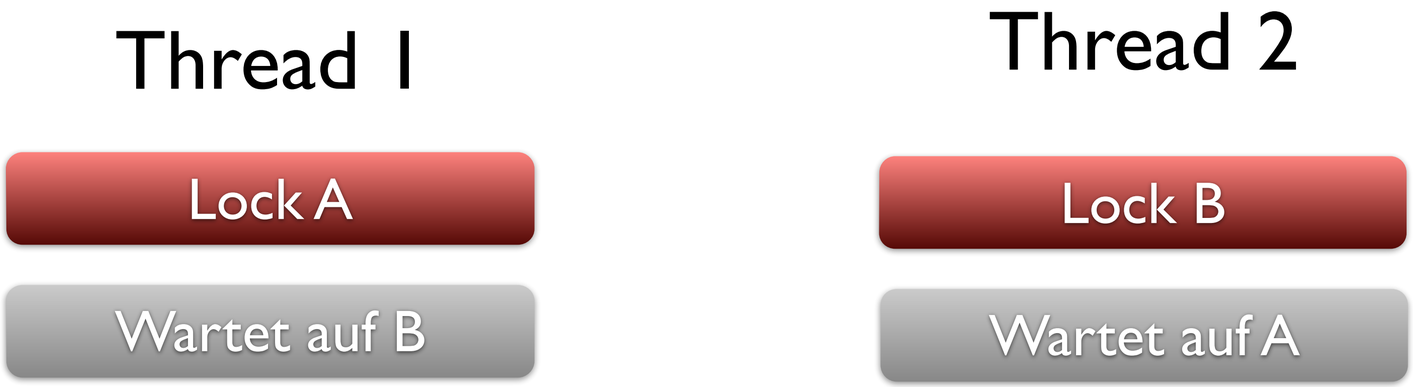
Race Conditions

- Gegenmaßnahmen:
  - Kopieren & Zusammenführen
    - Datenbank Transaktionen
    - Aufwändige Implementierung
    - Wird von CoreData verwendet
    - Zusammenführung bleibt ein Race

Race Conditions

- Gegenmaßnahmen:
  - Gegenseitiger Ausschluss (mutex, critical section)
    - Traditionell durch Locks
    - Blockieren von einzelnen Threads bei gleichzeitigem Zugriff auf kritische Stellen
    - Reduziert die kombinatorischen Möglichkeiten des Ablaufs
    - Gefahren: Wettstreit & Versperrung (Contention & Deadlock)

Deadlock



Race Conditions

- Gegenmaßnahmen:
  - Gegenseitiger Ausschluss durch GCD Dispatch Queues

Dispatch Queues

- Teil von Grand Central Dispatch (libdispatch)
- verbirgt Threads hinter anderer Abstraktion
  - Zerlegung der Arbeit in Blöcke, die in Queues auf ihre Abarbeitung warten
  - Threadpool wird automatisch verwaltet
- Concurrent & Serial

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

Dispatch Queues

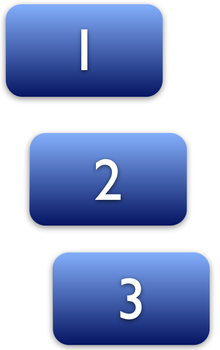
Concurrent Dispatch Queue

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

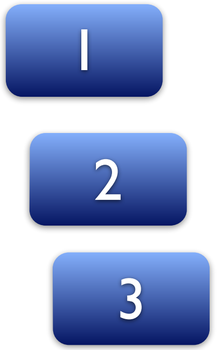


Dispatch Queues

Concurrent Dispatch Queue

Barrier
Block

Thread Pool

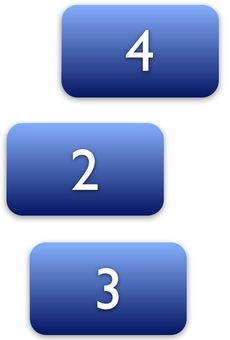


Dispatch Queues

Concurrent Dispatch Queue

Barrier
Block

Thread Pool



Dispatch Queues

Concurrent Dispatch Queue

Barrier Block

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

Barrier
Block

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

5

Barrier
Block

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

Barrier
Block

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

Thread Pool

Dispatch Queues

Concurrent Dispatch Queue

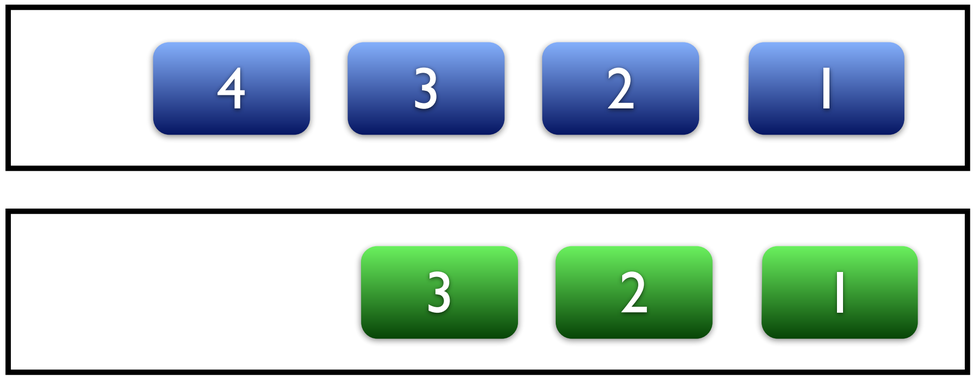
Thread Pool

Dispatch Queues

- Es gibt mehrere globale Dispatch Queues mit verschiedenen Grundprioritäten (Quality of Service).
- Barrier-Blocks können nur auf eigenen Concurrent Queues verwendet werden.

Dispatch Queues

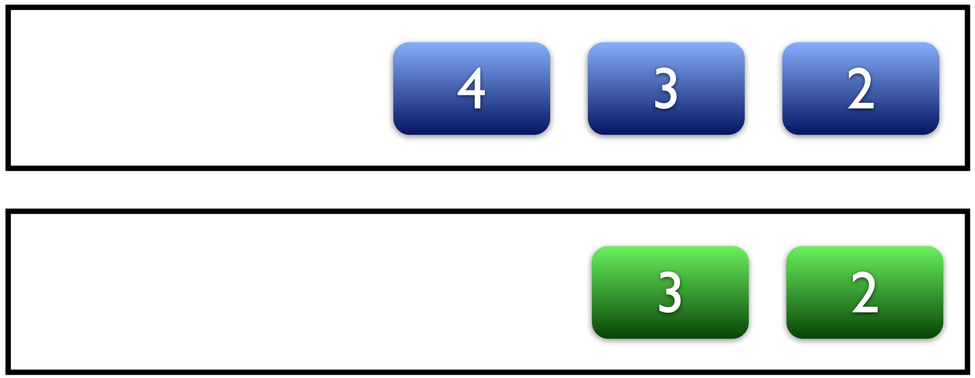
Serial Dispatch Queues



Concurrent Queue

Dispatch Queues

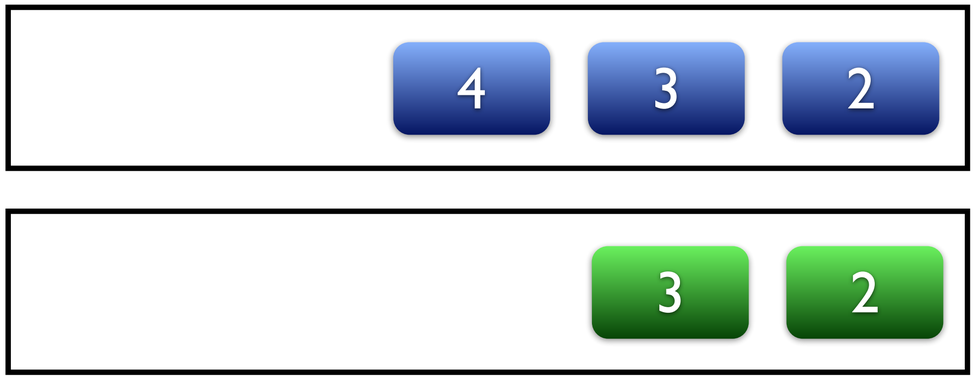
Serial Dispatch Queues



Concurrent Queue

Dispatch Queues

Serial Dispatch Queues



Concurrent Queue

Dispatch Queues

Serial Dispatch Queues

Concurrent Queue

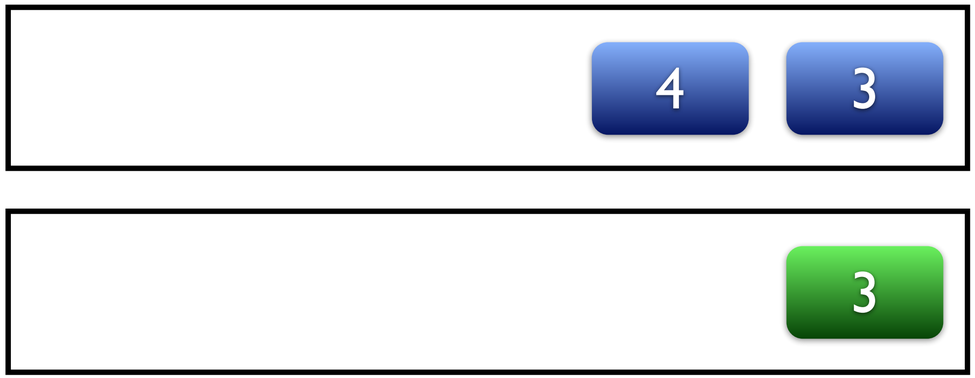
Dispatch Queues

Serial Dispatch Queues

Concurrent Queue

Dispatch Queues

Serial Dispatch Queues



Concurrent Queue

Serielle Dispatch Queues

„Inseln der Serialität in einem Meer von Gleichzeitigkeit“

Dispatch Queues

- dispatch_async
  - Gibt einen Block auf eine Queue und kehrt sofort zurück
- dispatch_sync
  - Blockiert den Aufrufer bis Block abgearbeitet wurde

Demo

Deadlock-Gefahr

- Auch dispatch_sync kann zu Deadlocks führen
  - Prinzipiell nur, wenn aus synchronen Blocks andere synchron-geschützte Ressourcen verwendet werden
- Erfahrungsgemäß schwer zu bändigen

The Problem with Threads

Edward A. Lee, 2006

www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf

The Problem with Threads

- Entwicklung des Ptolemy Project (in Java)
  - Aufwändiges Software-Engineering:
    - Code-Reife-Einstufungen (rot, gelb, grün, blau)
    - Design & Code Reviews von Experten
    - Nightly Builds & automatische Tests mit 100% Abdeckung
  - Wurde seit dem Jahr 2000 ohne Probleme breit genutzt…
  - …bis am 26. April 2004 ein Deadlock auftrat.

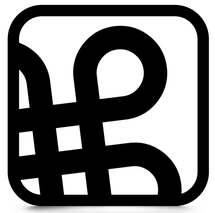
Demo

Zusammenfassung

- Wenn es nicht wirklich nötig ist, auf Nebenläufigkeit verzichten
- Immutable Objekte sind eine feine Sache
- Gemeinsame Daten mit Dispatch Queues schützen
- APIs so weit es sinnvoll geht asynchron entwerfen
- Code & Design Reviews durchführen
- Mit dem Schlimmsten rechnen!

Fragen?

Vielen Dank



Macoun