Rococo: Extract more concurrency from distributed transactions

Shuai Mu, Yang Cui, Yang Zhang,
Wyatt Lloyd, Jinyang Li
Tsinghua University, New York University,
University of Southern California, Facebook

What Large Web Sites Need

amazon

36.8 million sold/day

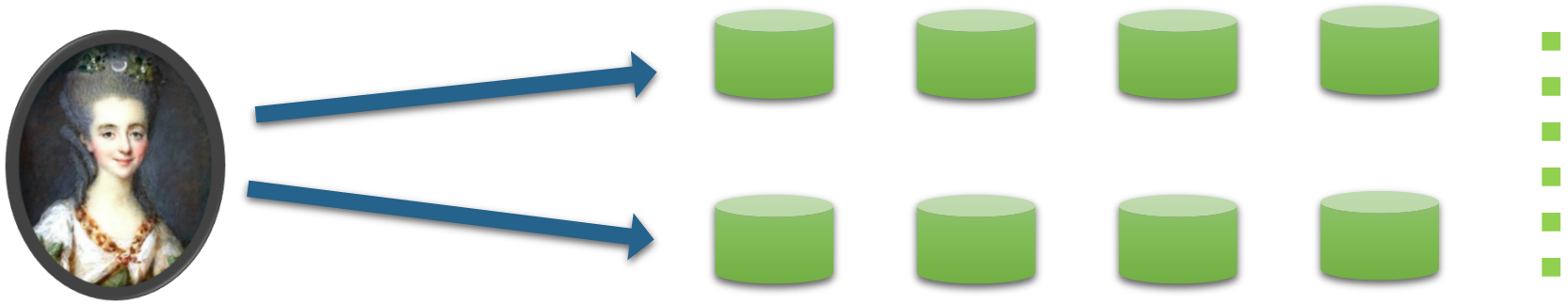
Alibaba

170 million sold/day

NYSE EURONEXT℠

$169 billion trade/day

Scalable Storage w/ Transactions!

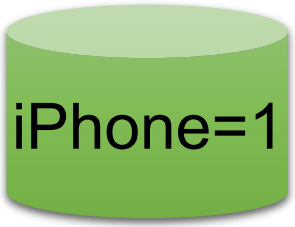


What is a Distributed Transaction?

```
BEGIN_TX
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
END_TX
```

Item	Stock
iPhone 6 Plus	1
iPhone Case	1

```
if (iphone > 0)  {
    iphone--;
}
```

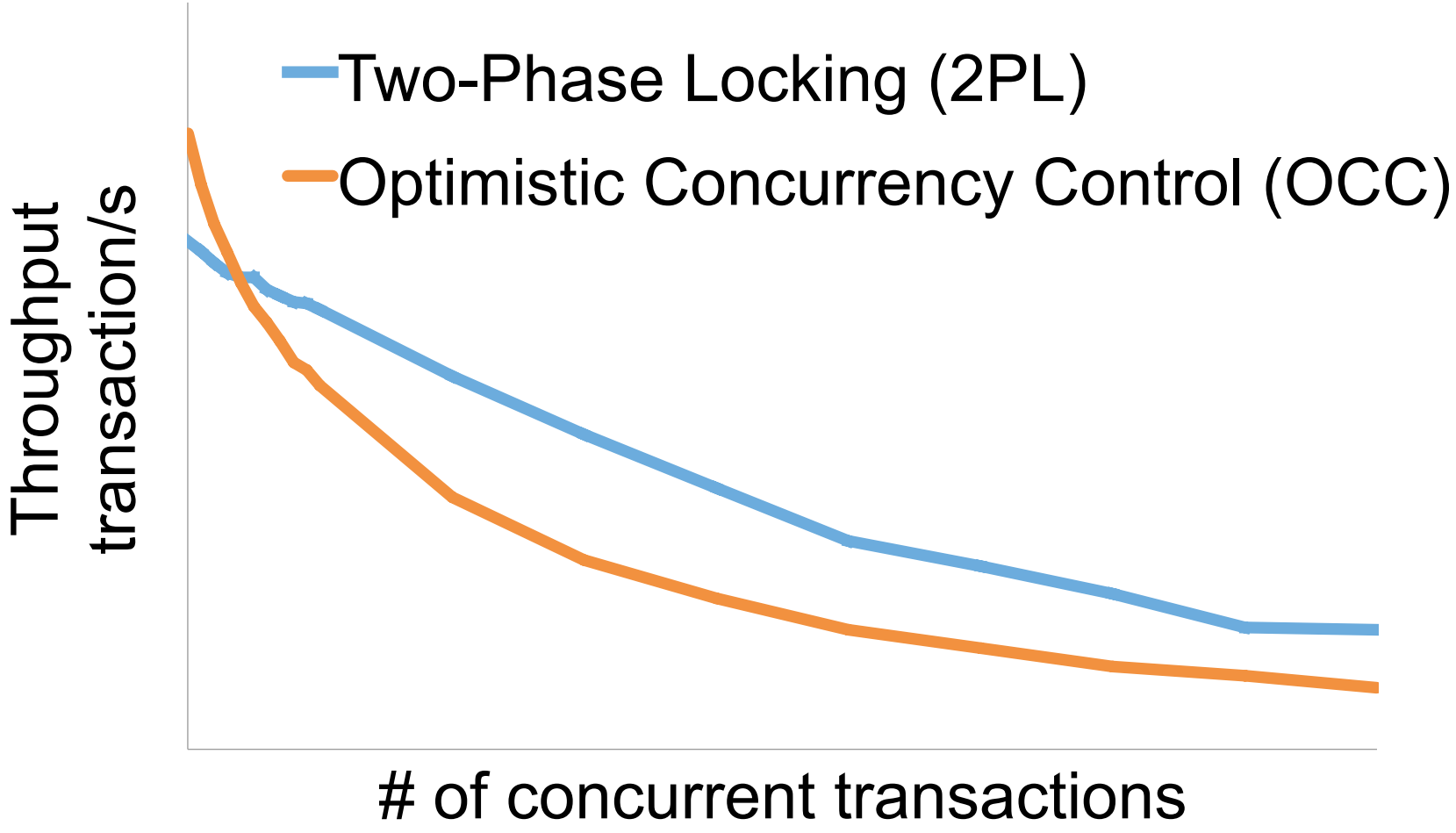



```
if (case > 0) {
    case--;
}
```

Transactions should be strictly serializable!
Otherwise…

Loss of serializability = angry customer

Serializability is Costly under Contention



OCC Aborts Contended Transactions

```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

Two-Phase Commit (2PC)

Execute

```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

```
If (iphone > 0)  {
    iphone--;
}
```

```
If (iphone >
    iphone--;
}
```

hone=1

Ca

```
if (case > 0) {
    case--;
```

```
ase > 0) {
    case--;
}
```

2PL Blocks Contended Transactions

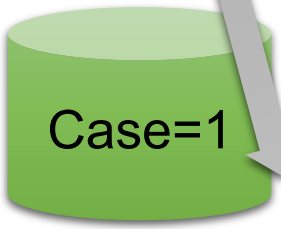
```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

2PC

Execute

```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

iPhone=1



Case=1

Achieve serializability w/o aborting or blocking*

* for common workloads

Rococo's Approach

```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

```
if (iphone > 0)  {
    iphone--;
}
if (case > 0) {
    case--;
}
```

```
If (iphone > 0)  {
    iphone--;
}
```

```
If (iphone > 0)  {
    iphone--;
}
```

iPhone=1

Case=1

```
if (case > 0) {
    case--;
}
```

```
if (case > 0) {
    case--;
}
```

Defer piece execution to enable reordering

Rococo Overview: Key techniques

1. Two-phase protocol
   - **Most** pieces are executed at the secon

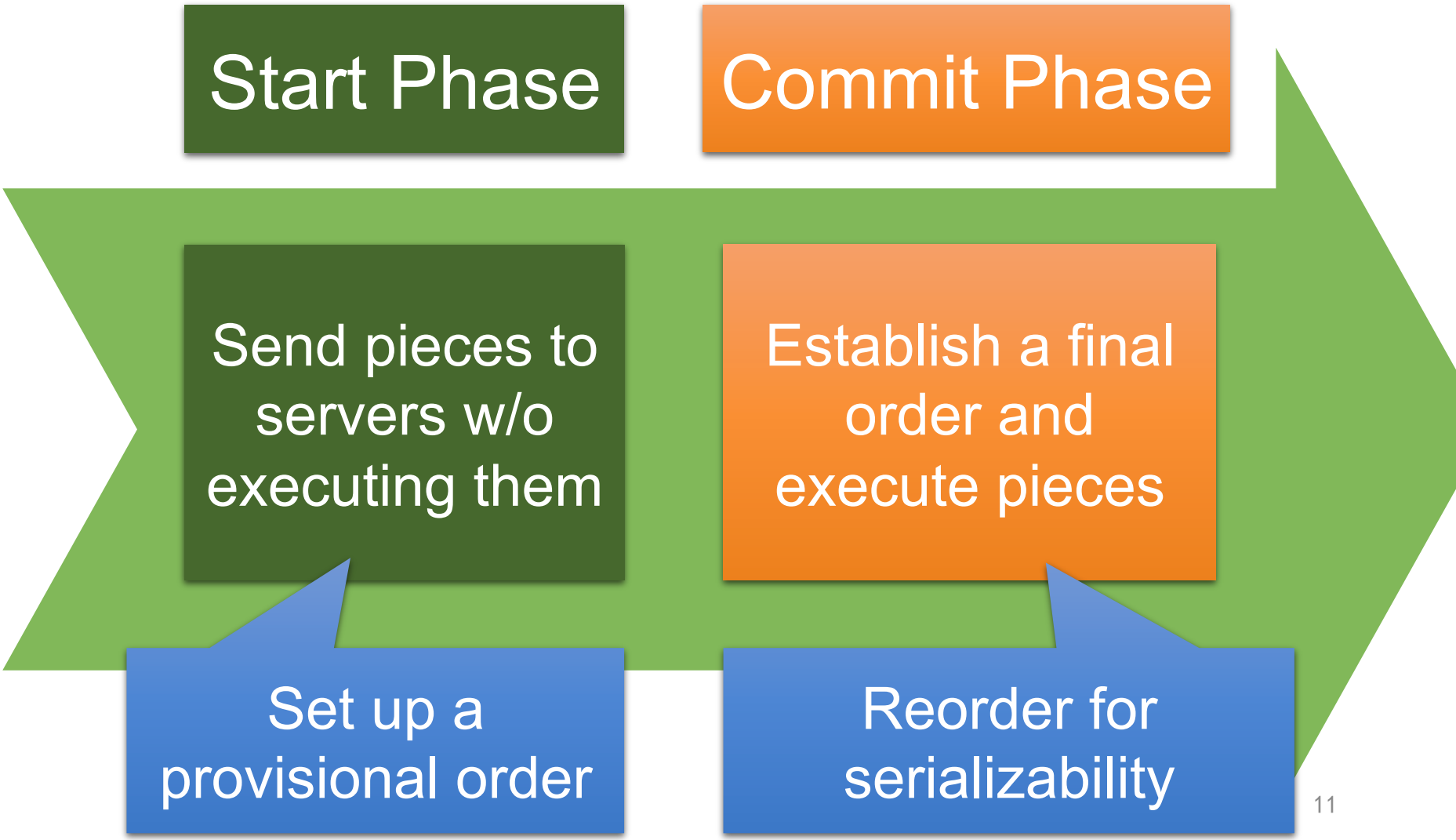
2. Decentralized dependency tracking
   - Servers track pieces' arrival order
   - Identify non-serializable orders
   - Deterministically reorder pieces

3. Offline workload checking
   - Identifies safe workloads (common)
   - Identifies small parts that need traditional approaches (rare)

Enable piece reordering for serializability

Avoid aborts for common workloads

#1 Two-phase protocol



#2 Dependency Tracking: Start Phase

T1:

if ...iphone--

if ... case--

dep

T2

T1

dep

T1

T2

T2:

if ... iphone--

if ... case--

if ... iphone--

if ... iphone--

dep

T1

T2

iPhone=1

if ... case--

if ... case--

dep

T2

T1

Case=1

#2 Dependency Tracking: Commit Phase

dep

T2

T1

dep

T1

Sort the cycle by any deterministic order

if ... iphone--

if ... iphone--

dep

T1

T2

T1

T1

T2

if ... case--

if ... case--

T1

T2

dep

T2

T1

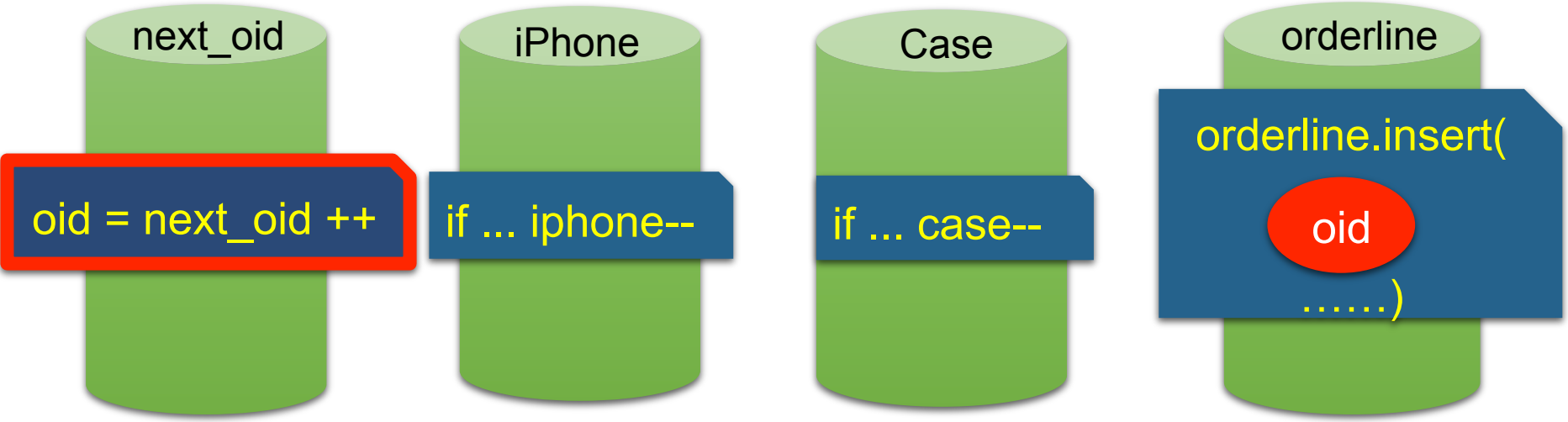
iPhone=1

Case=1

Problem: Not Every Piece is Deferrable

```
oid = next_oid ++
if ... iphone--
if ... case--
orderline.insert(oid, …)
```

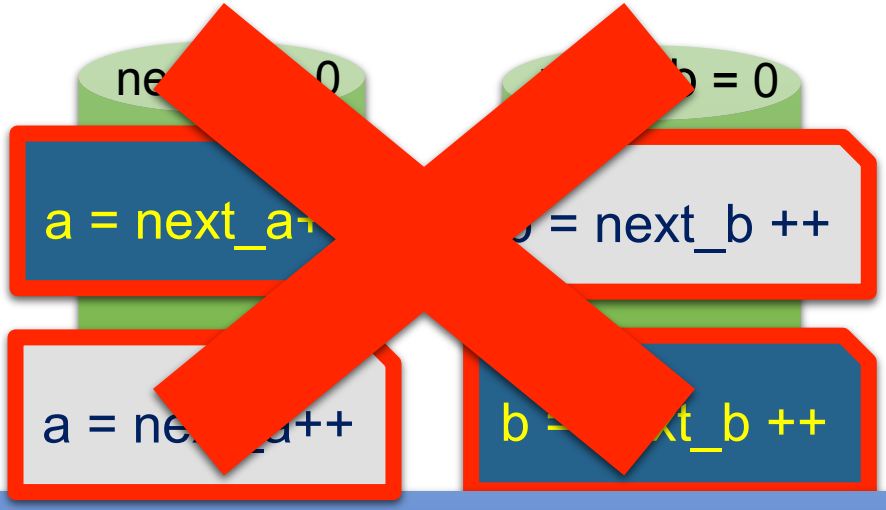
Intermediate Results Calls for Immediate Execution



Immediate Pieces are Naughty!

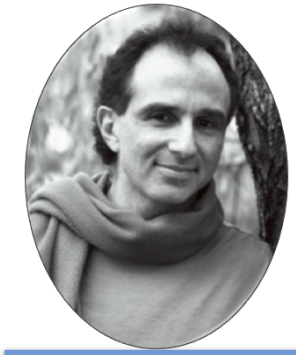
```
a = next_a++;
b = next_b++;
ol_a.insert(a, …);
ol_b.insert(b, …);
```

```
a = next_a++;
b = next_b++;
ol_a.insert(a, …);
ol_b.insert(b, …);
```



1. Common workloads have few immediate pieces
2. Pre-known workload → Offline check

#3: Offline Checking: Basic



T_1

An SC-cycle consists of both S-edge and C-edge

S(ibling)-edge linking pieces within a transaction

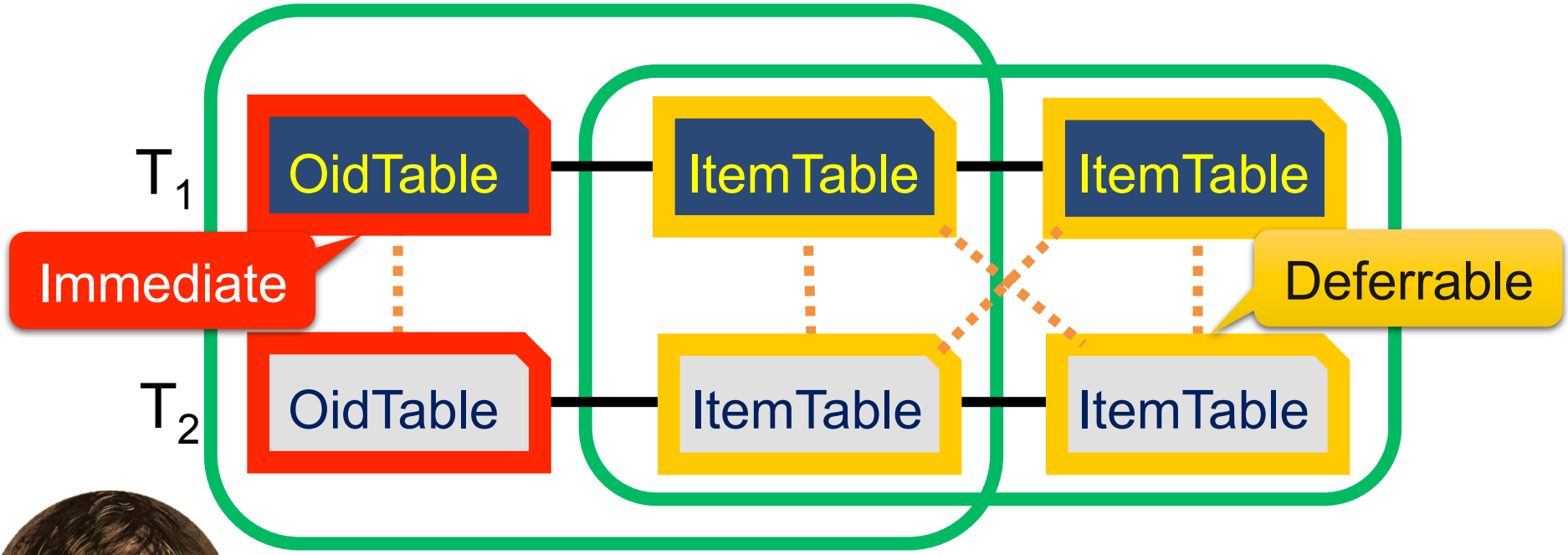
Item_Table — Item_Tab

Item_Table — Item_Tab

C(onflict)-edge linking pieces w/ conflicting access

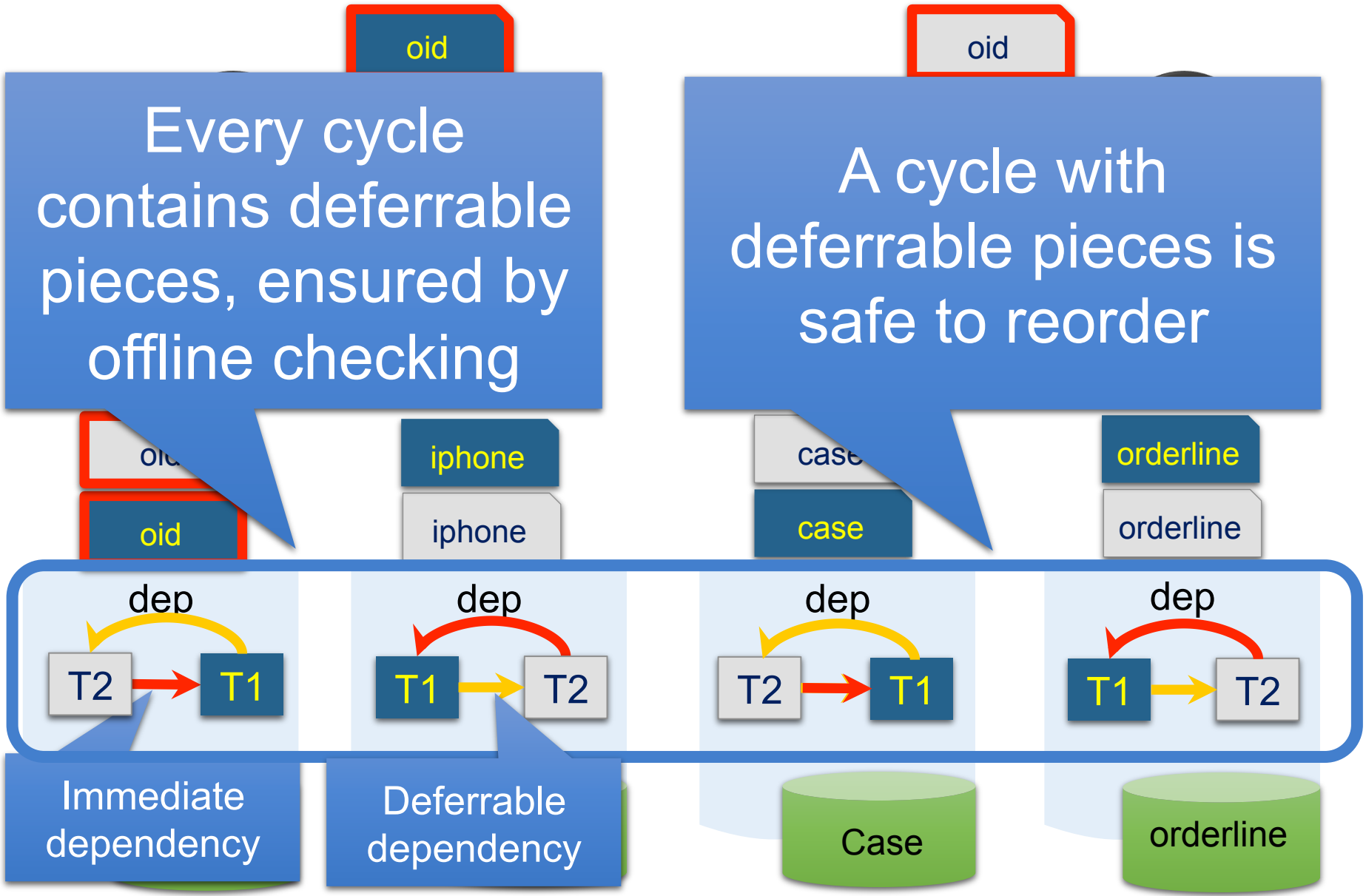
SC-cycles represent potential non-serializable executions that require 2PL/OCC

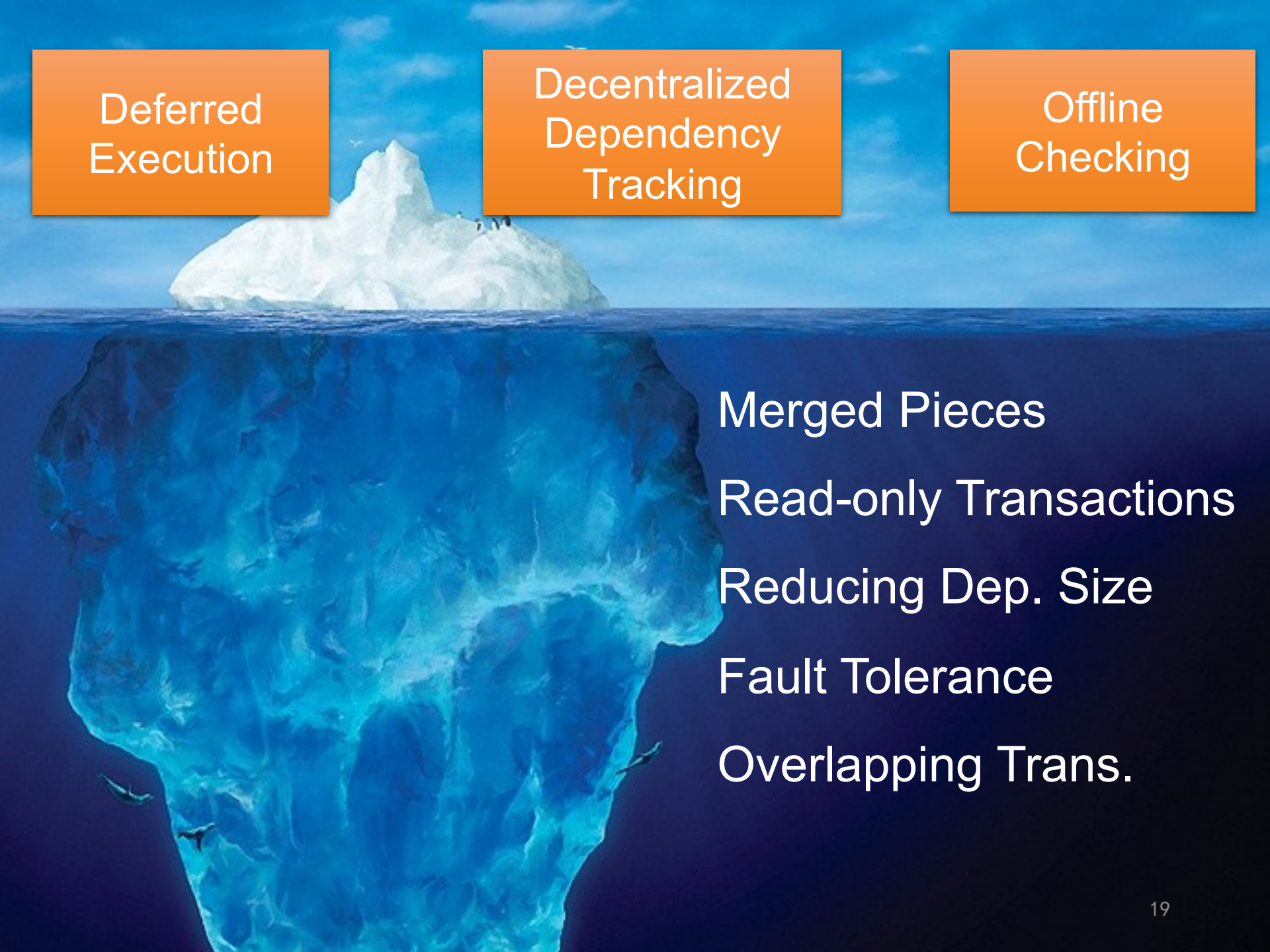
#3: Offline Checking: Enhanced



SC-cycles with deferrable pieces can be safely reordered!

Incorporating Immediate Pieces



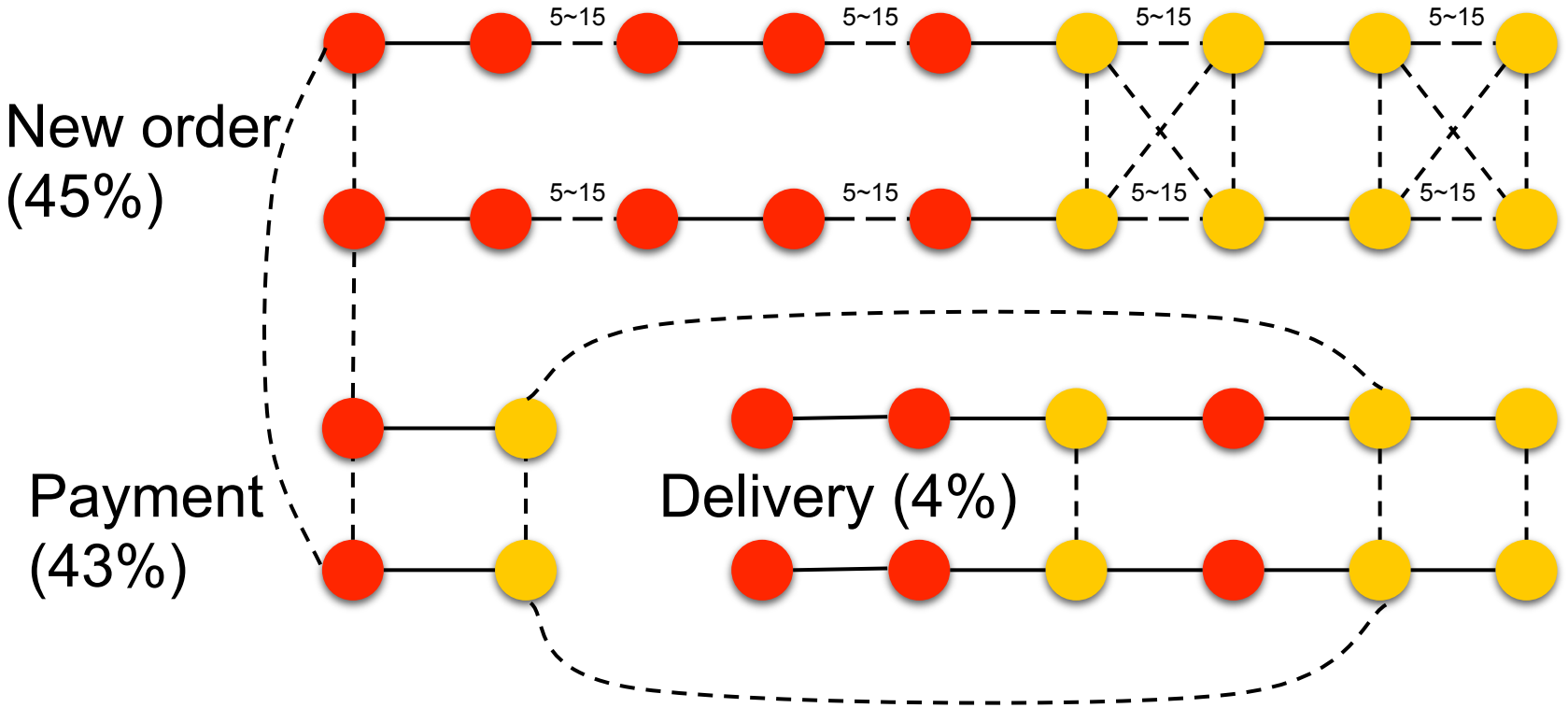

Deferred Execution
Decentralized Dependency Tracking
Offline Checking
Merged Pieces
Read-only Transactions
Reducing Dep. Size
Fault Tolerance
Overlapping Trans.

Evaluation

- How does Rococo perform under contention?
- How does Rococo scale?

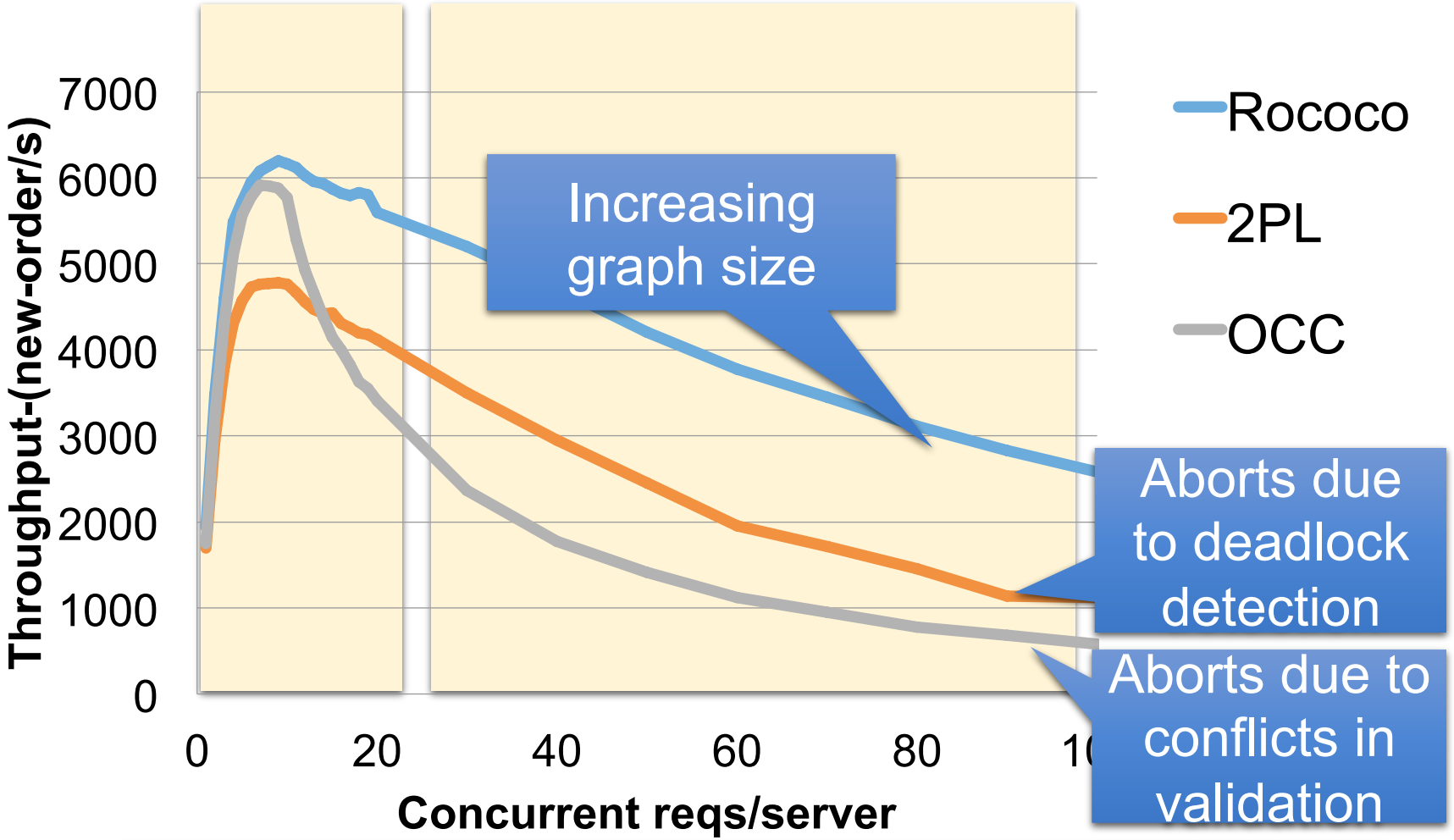
Workload: Scaled TPC-C

- One warehouse, many districts
- Partitioned by districts - all transactions distributed



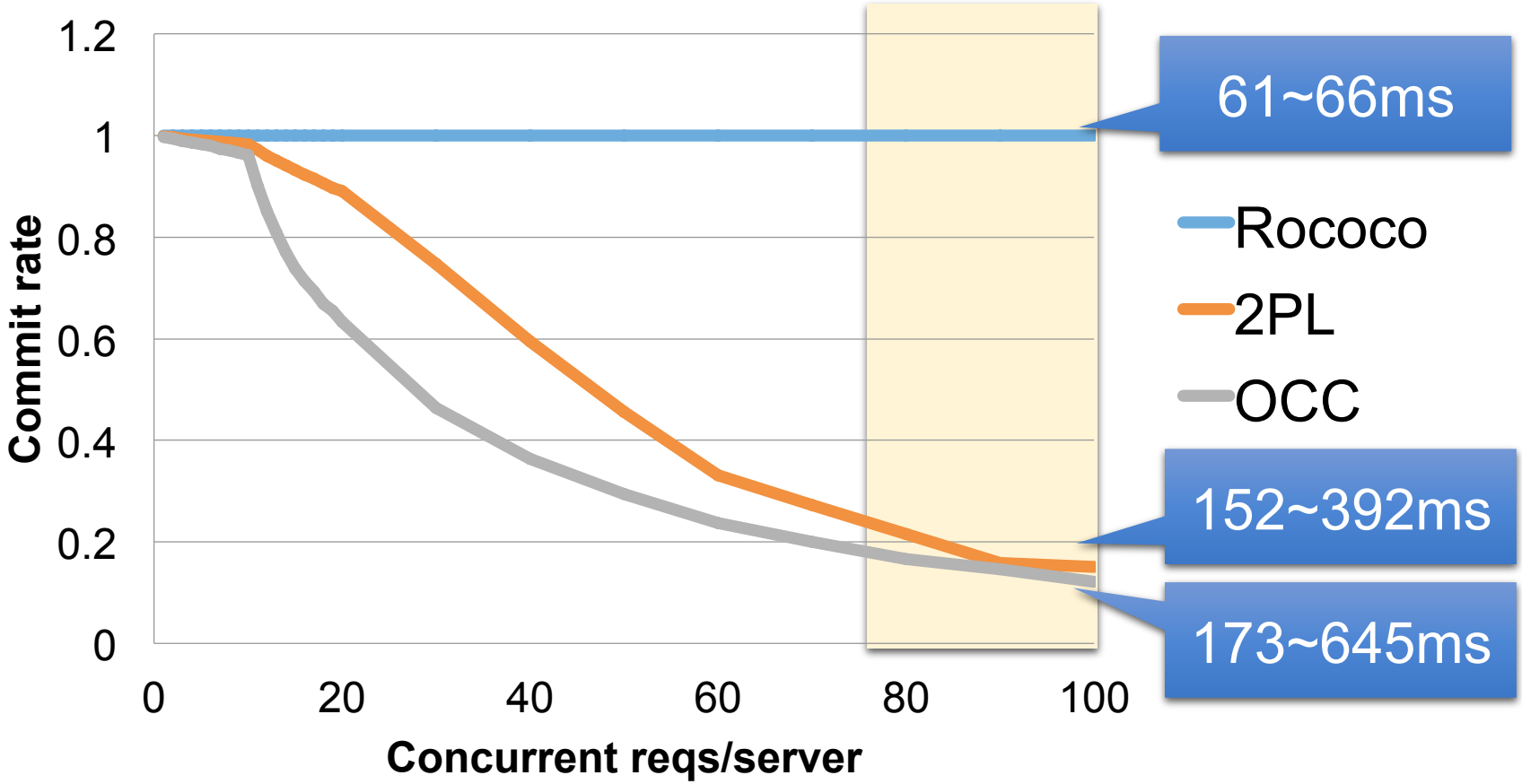
Read-only: OrderStatus(4%), StockLevel (4%)

Rococo Has Higher Throughput

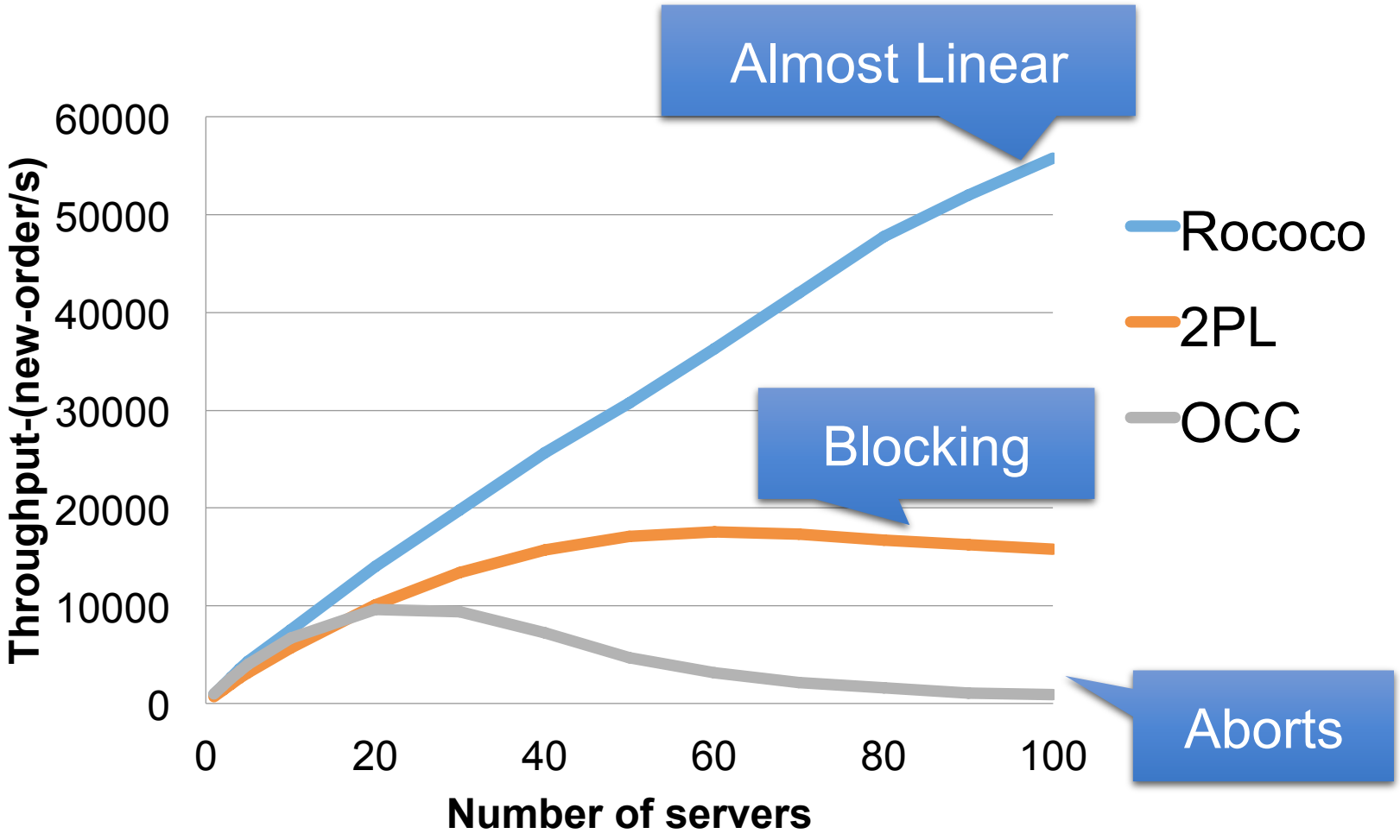


8 servers, 10 districts per server → Main source of contention is next_oid of each district

Rococo Avoids Aborts



Rococo Scales Out



10 districts per server, fixed # of items → contention grows slowly

Related Work

Decentralized dependency tracking

	Isolatio	
Rococo	Strict-Serial.	Rococo
Calvin [SIGMOD'12]	Strict-Serial.	Pre-ordered Locking
Spanner [OSDI'12]	Strict-Serial.	
HStore [VLDB'07]		
Lynx [SOSP'13]		
Granola [SIGMOD'1(		
Percolator [OSDI'10]	S.I.	OCC
Walter [SOSP'11]	P.S.I.	W-W Preclusion
COPS [SOSP'11]	Causal	Dependency Tracking

Centralized sequencing layer, difficult to scale

Conclusion

- Traditional protocols perform poorly w/ contention
  - OCC aborts & 2PL blocks
- Rococo defers execution to enable reordering
  - Strict serializability w/o aborting or blocking for common workloads
- Rococo outperforms 2PL & OCC
  - With growing contention
  - Scales out

Thank you!

Questions?

https://github.com/msmummy/rococo

Poster tonight!