Transactions

R&G Chapter 16

(slides adapted from content by J.Gehrke, J.Shanmugasundaram, and/or C.Koch)

Project 1 Graded - Grades available by Wednesday.
(along with HW1, HW2. 3-4 Graded Soon)

1- DO NOT MODIFY TEST CASE FILES
(please see me if you have, to avoid getting a 0)

2- Submit using tar/gz/zip. Please do not use rar.

3- Testing done with JVM/SDK 1.6
Please do not use 1.7/1.8 features
(Generics Type-Inference, Strings in Switches)

Sample TEAM file
(Makes grading scripts happy)

```
kirkj
spock32
mccoyle
```

One UBIT per line.
Nothing else!

Grading

Code Didn't Compile: Tentative 0
(See me)

Ran all provided RA Tests Successfully: 55 pts

Ran GB-Agg RA Tests Successfully: 5 pts

Ran all provided SQL Tests Successfully: 40 pts

Ran GB-Agg SQL Tests Successfully: +5 pts

Submitted to Original Deadline: +20 pts

Of projects that successfully compiled

Min: 35.0

Max: 119.2

Avg: 91.4 (+/- 24.9)

Reminder: Midterm on Monday

Covers Material Up to and Including Optimization

See Syllabus for Exact Chapter/Section #s

Time to get up into the guts of a database

How do databases keep data moving at warp speed?

Database Workloads

- Two major interactive database workloads:
  - OLAP: Online Analytical Processing
    - Big queries about large, static data.
  - OLTP: Online Transaction Processing
    - Many queries and updates on small data.

OLTP Workloads

- Small data manipulations: Lots of IOs.
  - Keep the disk active by running many tasks in parallel.
  - Keep latencies low by making progress on multiple tasks at once.
- It is the DBMS's responsibility to interleave parallel tasks correctly.

Transactions

What does it mean for a database operation to be correct?

The database doesn't know anything about the business logic. It doesn't know how the user/ calling application is using data. All it knows is what values have been read and which values have been written.
We need to develop a notion of "correctness" relative to this view of the data -- reads and writes.

Transactions

What does it mean for a database operation to be correct?

How does a database interact with its users?

The database doesn't know anything about the business logic. It doesn't know how the user/calling application is using data. All it knows is what values have been read and which values have been written.
We need to develop a notion of "correctness" relative to this view of the data -- reads and writes.

Transactions

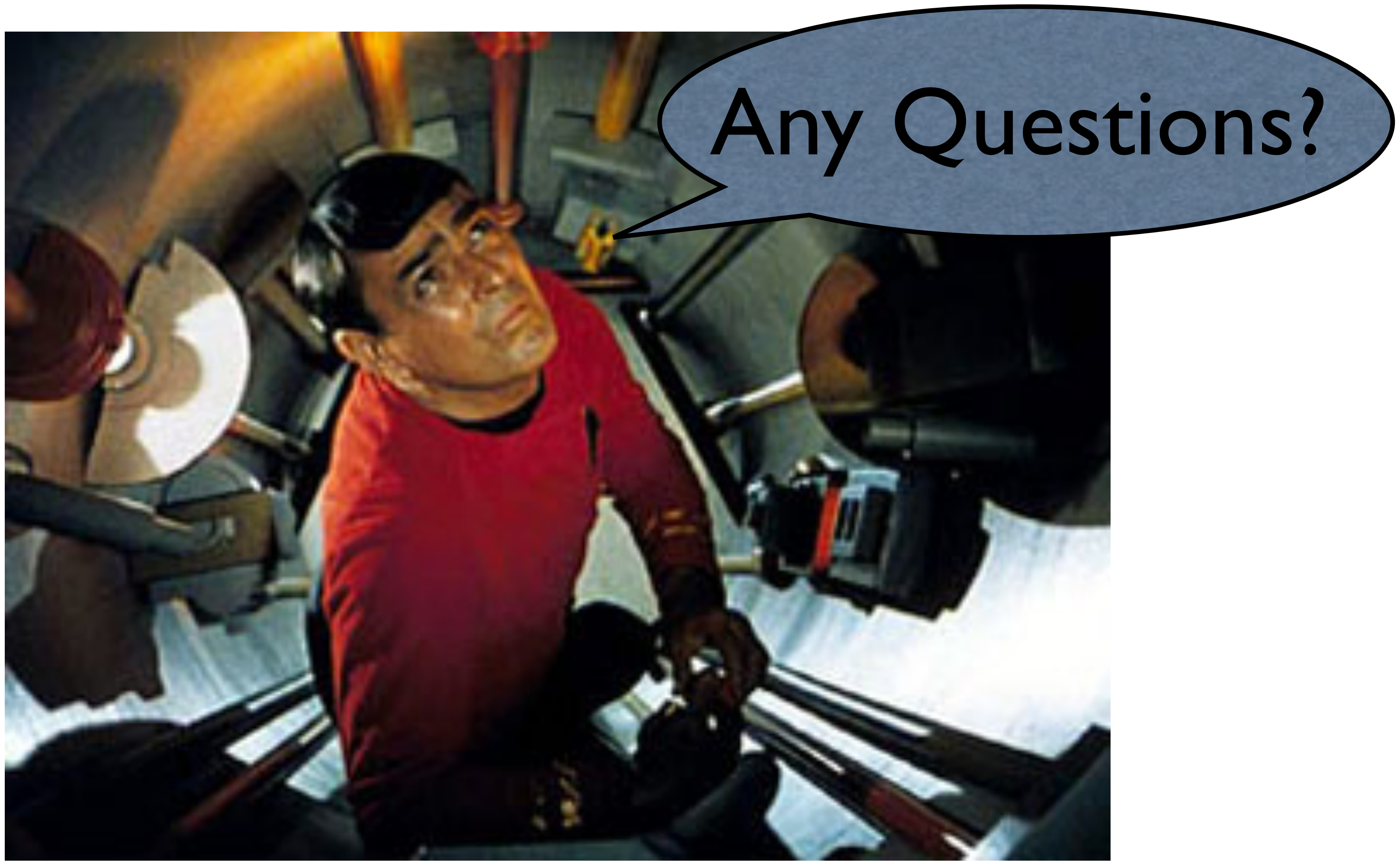
- Users group sequences of interactions with the DBMS into a Transaction.
  - *Guarantee*: From the user's perspective, transactions execute fully and on their own.
  - *Guarantee*: Transactions preserve data consistency (as per Integrity Constraints).
- The DBMS interleaves DB operations while respecting the above guarantees.

Transactions

- Challenge 1: Interleaving
  - How do we negate the effects of interleaving operations from two parallel transactions?
- Challenge 2: Crashes
  - How do we avoid leaving the system in an inconsistent state after a crash?

Atomicity

- A transaction completes by committing, or terminates by aborting.
    - Logging is used to undo aborted transactions.
- **Atomicity**: A transaction is (or appears as if it were) applied in one 'step', independent of other transactions.
    - All ops in a transaction commit or abort together.
    - All other transactions commit 'logically' in a fixed order. (Serializability)


Any Questions?

Example

```
T1: BEGIN A=A+100,  B=B-100  END
T2: BEGIN A=1.06*A, B=1.06*B END
```

- Intuitively, T1 transfers $100 from A to B and T2 credits both accounts with interest.
- What are possible interleaving errors?

T2.1, T1.1, T1.2, T2.2 -> Apply interest before crediting A, and after debiting B; Bank saves 12 bucks
T1.1, T2.1, T2.2, T1.2 -> Apply interest after crediting A, and before debiting B; Bank loses 12 bucks

Example: Schedule

Time	T1	T2
	`A=A+100`	
		`A=1.06*A`
	`B=B-100`	
		`B=1.06*B`

A schedule is an interleaving of operations

Example: Schedule

Time	T1	T2
	A=A+100	
		A=1.06*A
	B=B-100	
		B=1.06*B

OK!

A schedule is an interleaving of operations

Example: Schedule

Time	T1	T2
	`A=A+100`	
		`A=1.06*A`
		`B=1.06*B`
	`B=B-100`	

Friday, March 1, 13

Ordering is bad. Bank loses 12 bucks.

Example: Schedule

Time	T1	T2
	`A=A+100`	
		`A=1.06*A`
		`B=1.06*B`
	`B=B-100`	

Not OK!

Ordering is bad. Bank loses 12 bucks.

Example: The DBMS's View

Time	T1	T2
↓	`R(A)`	
	`W(A)`	
		`R(A)`
		`W(A)`
		`R(B)`
		`W(B)`
	`R(B)`	
	`W(B)`	

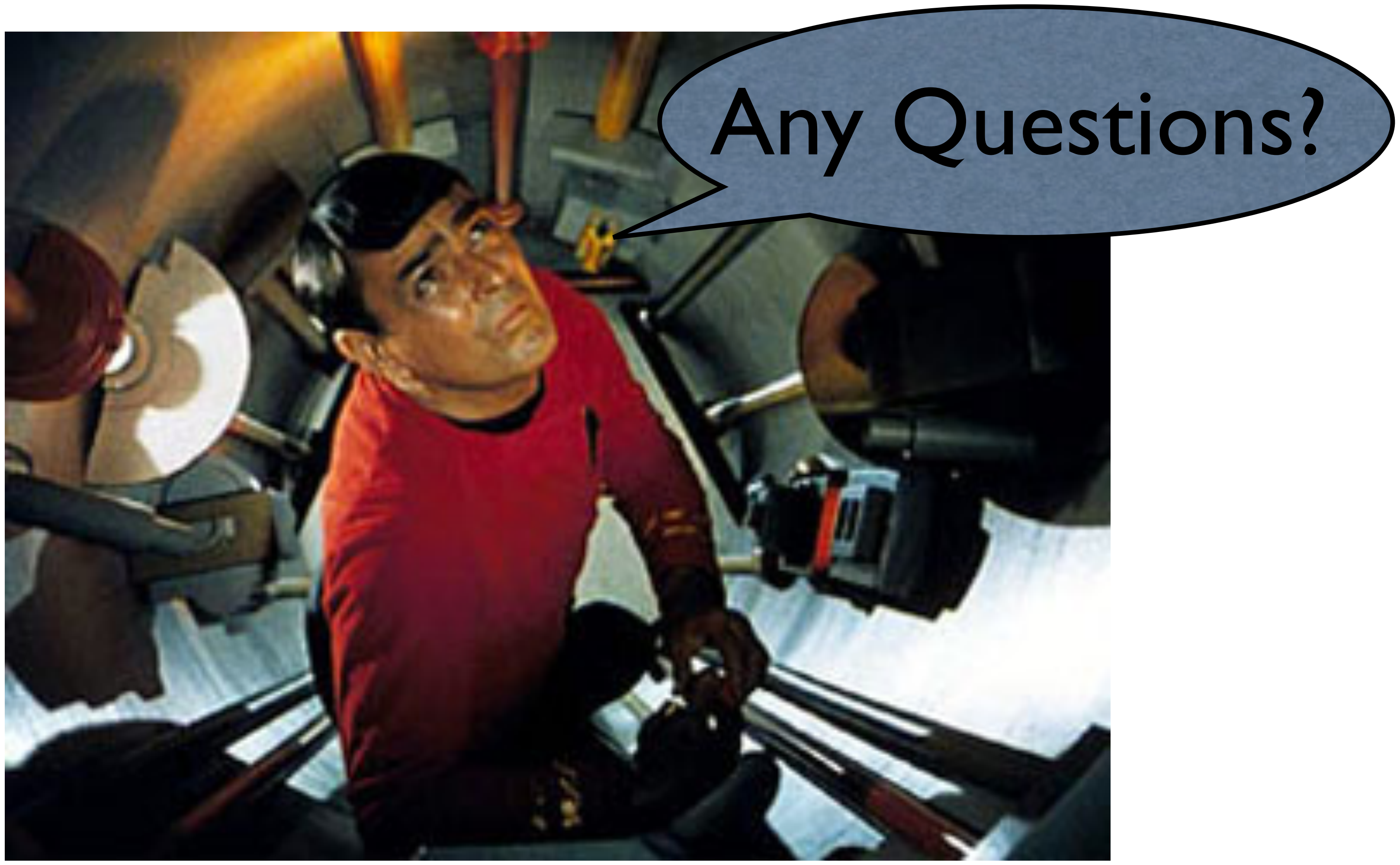
Ordering is bad. Bank loses 12 bucks.

Example: The DBMS's View

Time	T1	T2
↓	R(A)	
	W(A)	
		R(A)
		W(A)
		R(B)
		W(B)
	R(B)	
	W(B)	

Not OK!

Ordering is bad. Bank loses 12 bucks.


Any Questions?

Scheduling

- **Serial Schedule**: A schedule that does not interleave the actions of different transactions.
- (Two) **Equivalent Schedules**: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second.
- **Serializable Schedule**: A schedule that is equivalent to some serial execution of the transactions.
  - If the transactions preserve consistency, every serializable schedule does too.

Interleaving Anomalies

- Reading uncommitted data (write-read/WR conflicts; aka “dirty reads”)

```
T1: R(A),W(A),                        R(B),W(B),ABRT
T2:                R(A),W(A),CMT,
```

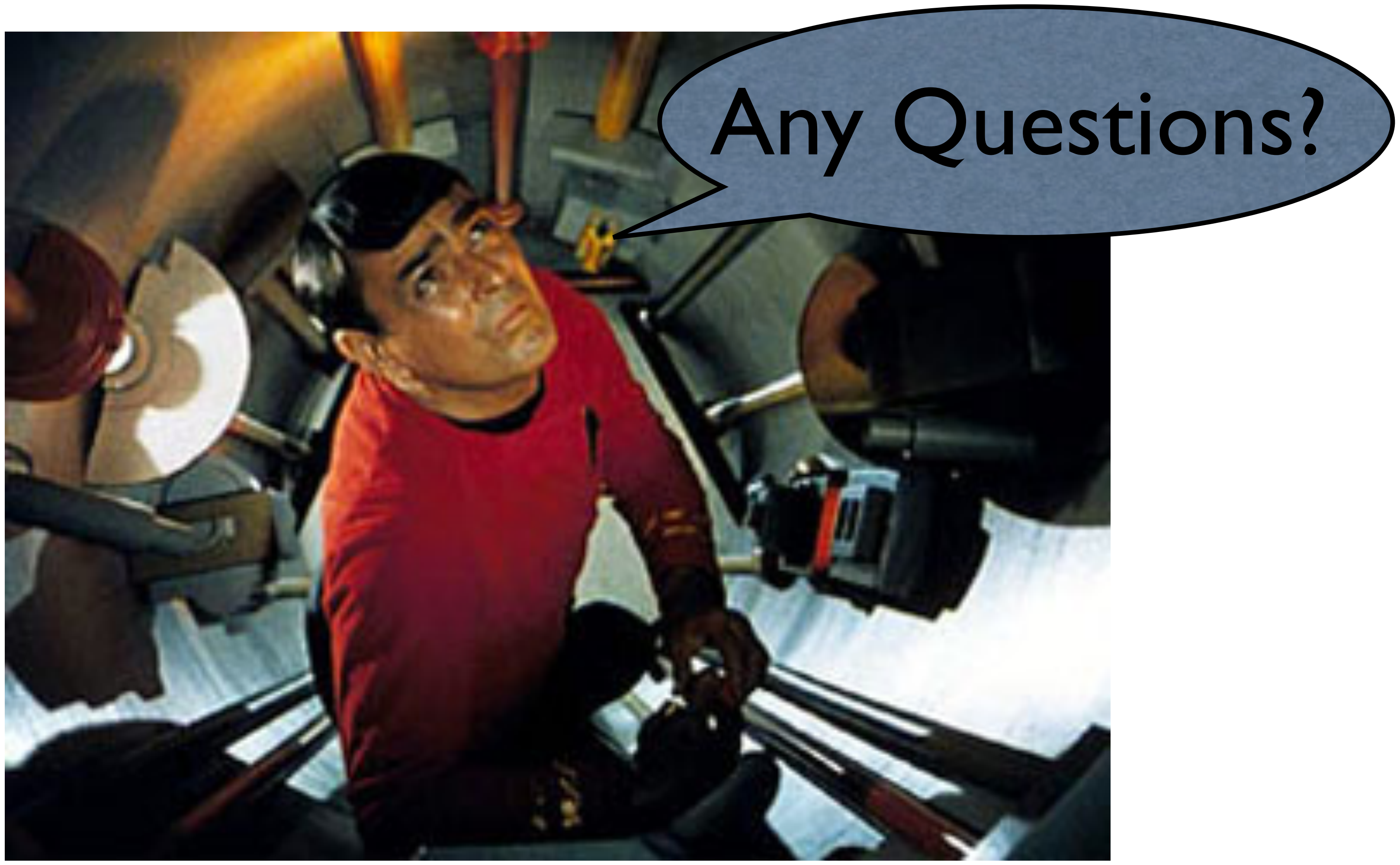
- Unrepeatable Reads (read-write/RW conflicts)

```
T1: R(A),                   R(A),W(A),CMT
T2:          R(A),W(A),CMT,
```

Interleaving Anomalies

- Overwriting Uncommitted Data (write-write/WW Conflicts)

```
T1: W(A),                  W(B),CMT
T2:      W(A),W(B),CMT,
```


Any Questions?

Lock-Based Concurrency

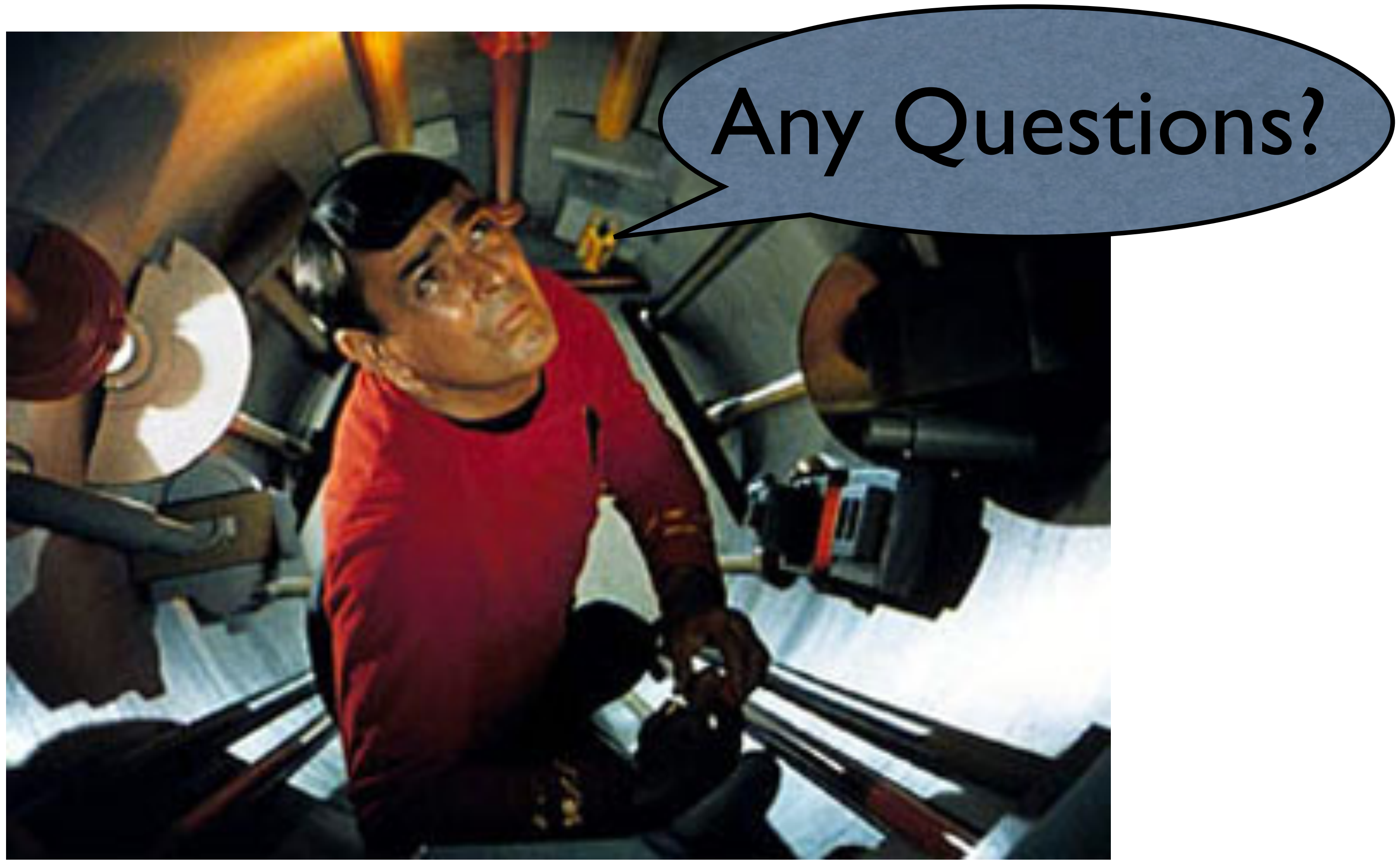
- Strict Two-Phase Locking
  - Before reading, obtain a S(hared) lock
  - Before writing, obtain a X(clusive) lock
  - Hold locks until transaction completes
- Why two-phase locking?

Phase 1: Acquire, Phase 2: Release
Why? :
1 - Holding the lock until the transaction ends allows only strict serializability (e.g., no WW/WR conflicts)
2 - Makes it easier to support aborts (modified data is never read from or overwritten)

Lock-Based Concurrency

- Non-Strict 2PL Variant
  - Release locks anytime, but can't acquire locks after first release.
- General 2PL issues
  - What can go wrong with locking?
  - What are some ways around these issues?

Locking opens up the possibility of deadlock. There are a number of ways around this, but the general strategy is to either run a deadlock detecter (track dependency chains, and ID loops), force xacts to pre-declare all data that they plan to modify/write, or enforce a deterministic ordering of locking requests.


Any Questions?

Aborting Transactions

- If a transaction T_i is aborted, all of its actions have to be undone!
  - What if T_j reads a value modified by T_i?
  - How can we prevent this from happening?
- The DBMS maintains a log of every write, which it uses to undo aborted xacts.
  - The log also assists in crash recovery!

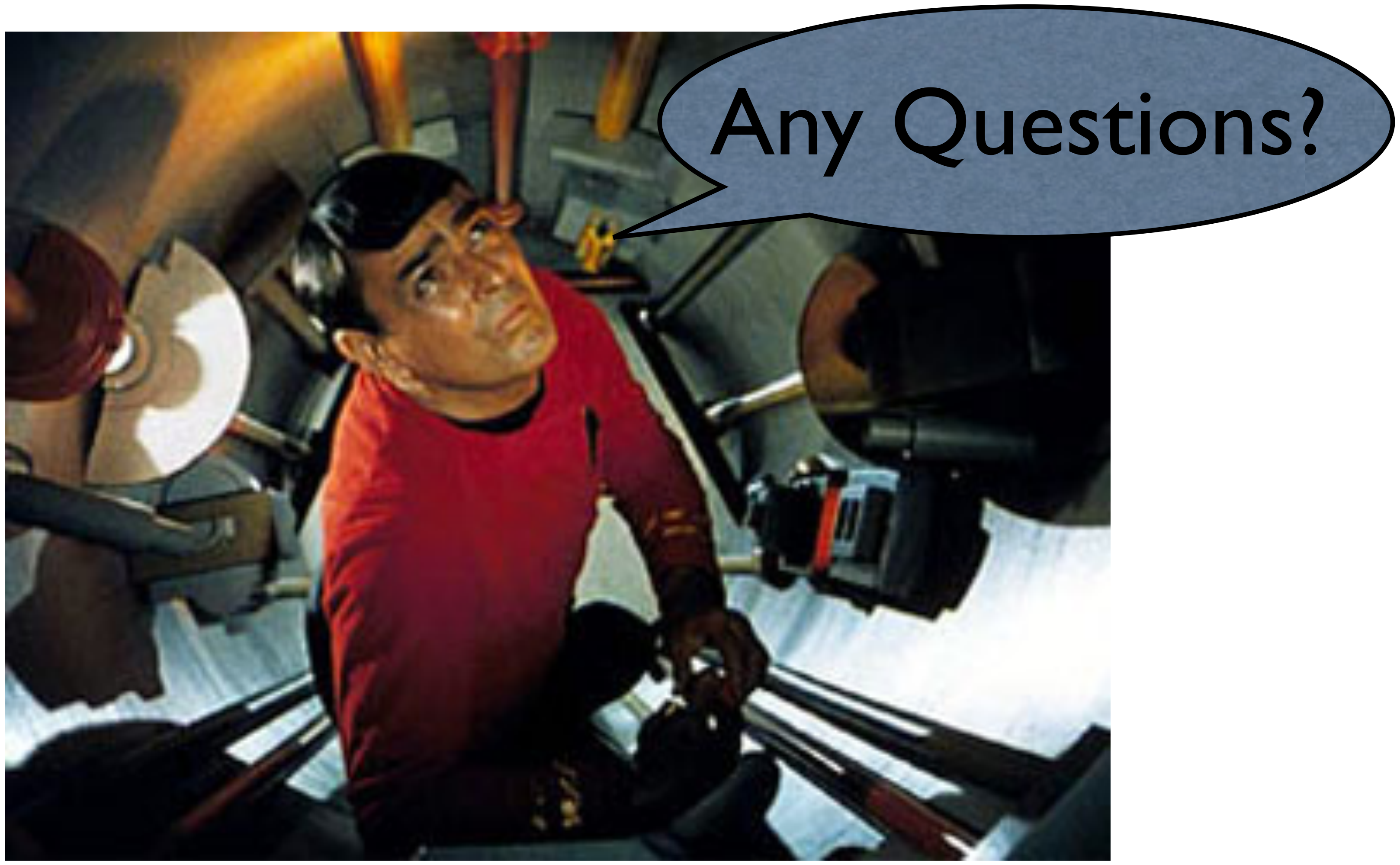
If Tj reads a value, its read is now invalid, and it must be retried. This is typically done by aborting the transaction and re-issuing it.
We can prevent this by using strict 2PL, or by “hiding” Ti’s writes from Tj until Ti commits

The Log

- The log records:
  - T_i writes: Both old and new values of object.
    - Log record must make it to disk before the changed data page!
  - T_i commits/aborts: A record of the event.
- Log records are chained (stored as a linked list) for each xact, so it's easy to undo an xact.

The Log

- DBMSes often store redundant duplicates of the log on stable storage.
- All log-related activities (and locking/deadlock detection) are handled transparently by the DBMS.


Any Questions?

Transaction Variants

- Cross-server transactions.
    - Multiple DBMS servers participate.
- Non-interactive transactions.
    - Specify full transactions programatically.
- Functionally-limited transactions.
    - Read-only, No side-effects, Limited writes

Crash Recovery Preview

- The ARIES recovery algorithm (3 Phases)
  - **Analysis**: Scan through the log to identify all xacts active at time of crash and all dirty pages in the buffer pool.
  - **Redo**: Redo all updates to dirty pages in the buffer pool to ensure that logged updates are carried out.
  - **Undo**: Use 'before' value from log to cancel out the writes of all xacts that were active at the time of crash. (need to guard vs crashing during recovery)

Summary

- Concurrency control is one of the most important features provided by a DBMS.
- Users do not need to worry about concurrency issues.
  - DBMS automatically inserts locking/scheduling requests as needed.
- Write-ahead logging is used to undo the actions of aborted transactions.
- Consistent State: Only the effects of committed xacts are seen by other xacts.