The Timing and Cost of Choices

Hubert Matthews, Oxyware Ltd
hubert@oxyware.com
http://www.oxyware.com
ACCU Spring Conference 2003

Choices are everywhere

- We make design choices all the time
- We think long and hard about those choices
  - But how much do we think about the choice process itself?
- Are there any patterns in the choice process?
- How do process and product choices interact?
- What is the relation to time in all this?

Copyright © 2003, Oxyware Ltd

Choose, Check & Use pattern

- Make a choice
- Validate or test that choice (optional)
- Use the choice (deliver service)

Copyright © 2003, Oxyware Ltd

Ordering ice cream - late check

- Late check (at point of delivery)
- Possible delayed failures
  - May require extra use cases (“Spam is off”)
  - Diagnostic distance - context of choice is remote

Copyright © 2003, Oxyware Ltd

Ordering ice cream - early check

- Early check avoids delayed failure
- But waitress's "cache" may be out of date
  - Time-of-check to time-of-use (TOCTOU)
  - Business error, not a technical error - assumption

Copyright © 2003, Oxyware Ltd

Late binding addiction

- General trend towards later binding
  - Scripting languages, reflection, configurability
- Most GoF patterns introduce later binding
- "All problems in computing can be solved with an extra layer of indirection"
  - Apart from having too many layers of indirection
- So what are the costs of later binding?
- Why do we ever use early binding?

Copyright © 2003, Oxyware Ltd

Early binding addiction

- Template metaprogramming
- Assembler
- Waterfall process - big up-front design
- Point solutions
  - Fix today's problems now, hard-wired approach
- Can allow forward movement on "stuck" projects
  - Reduces analysis paralysis
  - Chipping away at the problem

Copyright © 2003, Oxyware Ltd

Early v. late binding

Early	Late
Performance (not always!)	Flexibility
Certainty	Reuse
TOCTOU	Delayed failure
Monolithic	Metadata + carrying cost
	Configuration overhead

Copyright © 2003, Oxyware Ltd

Examples

	Code	Business rules	Data format	Reqts	Call
Early	Statically linked, no reuse, type safety	Hard-coded, not bypassable	Compact, fast, inextensible	Waterfall, TOCTOU	Template, overload
Late	DLLs, eval, symbol table, mv/cp, interpreter, search path	Type Object, stored procs	XML, format failure, data dictionary	Agile process, refactor, tests	Virtual, vtable, null pointer, config

Vendor selection, classic Strategy pattern, copy-on-write strings, bounced cheque, defensive programming, …

Copyright © 2003, Oxyware Ltd

Binding != coupling

- Binding time is about when choices are made
- Coupling is about the amount of shared knowledge between interacting parties
  - Richness of interface/protocol

	Loosely coupled	Tightly coupled
Early bind	C++ templates	Design by contract
Late bind	Smalltalk calls	SMTP client and server

Copyright © 2003, Oxyware Ltd

Prevention, removal, tolerance

- Prevention of injection (choose)
- Removal (check)
- Mask or fail (use)

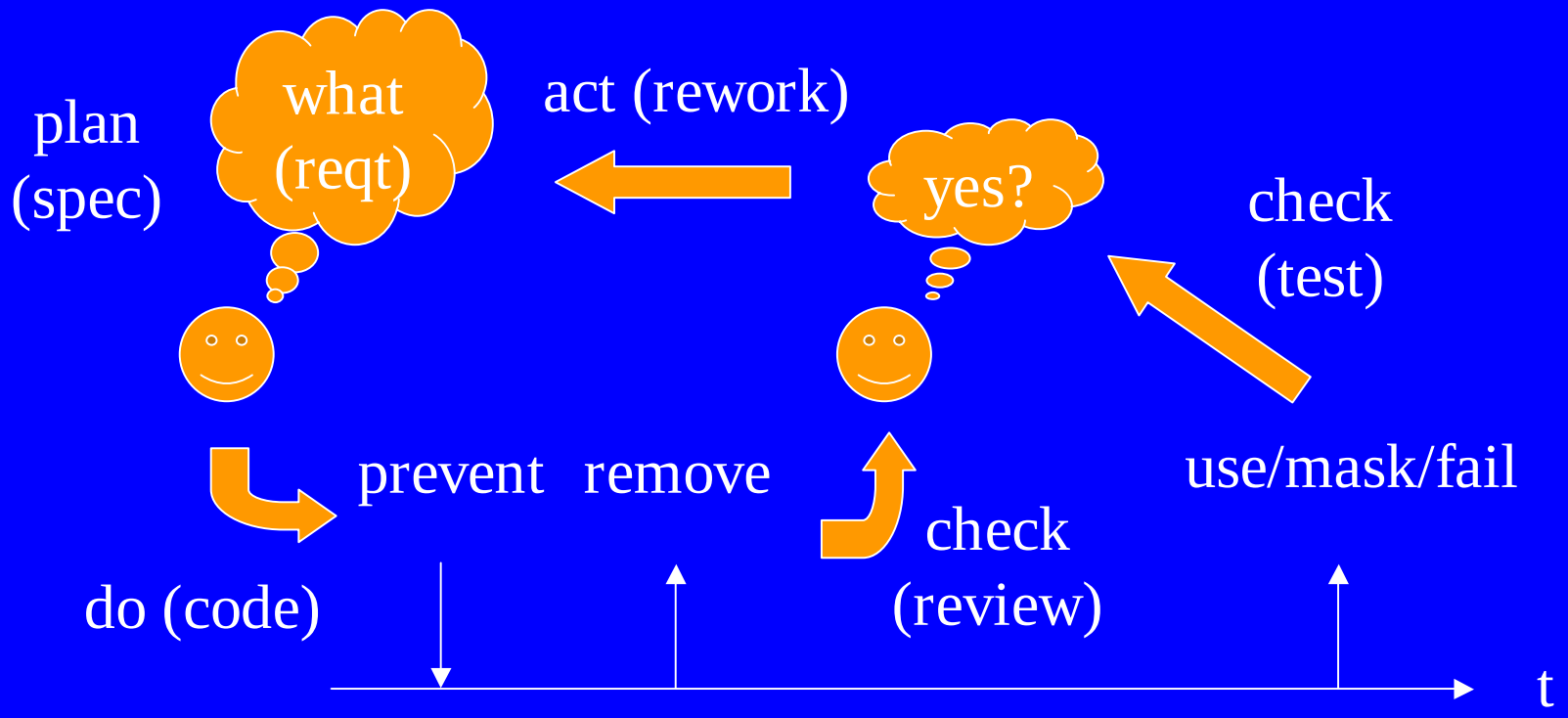
Copyright © 2003, Oxyware Ltd

Intent v. mechanism

- Early bind failure (at point of use) is failure of intent (TOCTOU)
  - Everything worked fine but it's not what was wanted
- Late bind failure often manifests itself as a technical error (exception)
  - Failure of mechanism: file not found, etc
- How can we reduce errors of intent?
  - And what about errors of mechanism?

Copyright © 2003, Oxyware Ltd

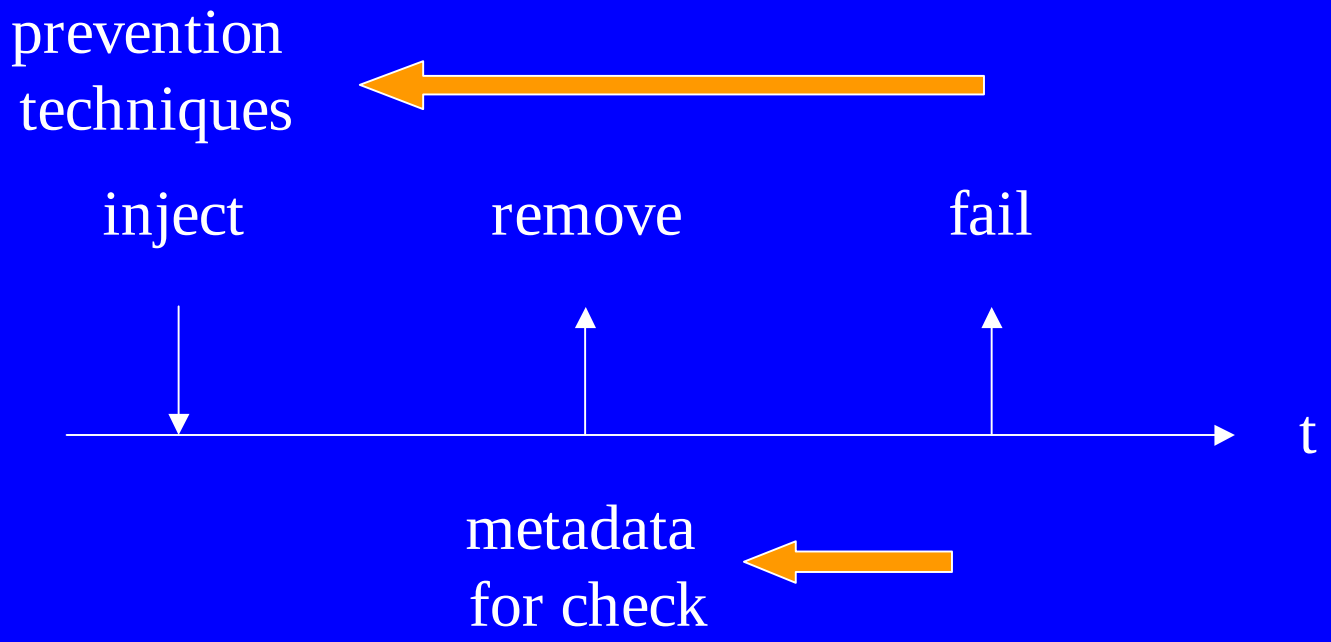
Norman & Deming - intent



- Gulf of execution and gulf of evaluation

Copyright © 2003, Oxyware Ltd

Product to process feedback



- Sometimes feedback is too expensive
  - Safety-critical systems, hazard lists, FMEA, etc
  - Feed forward estimates must be used instead

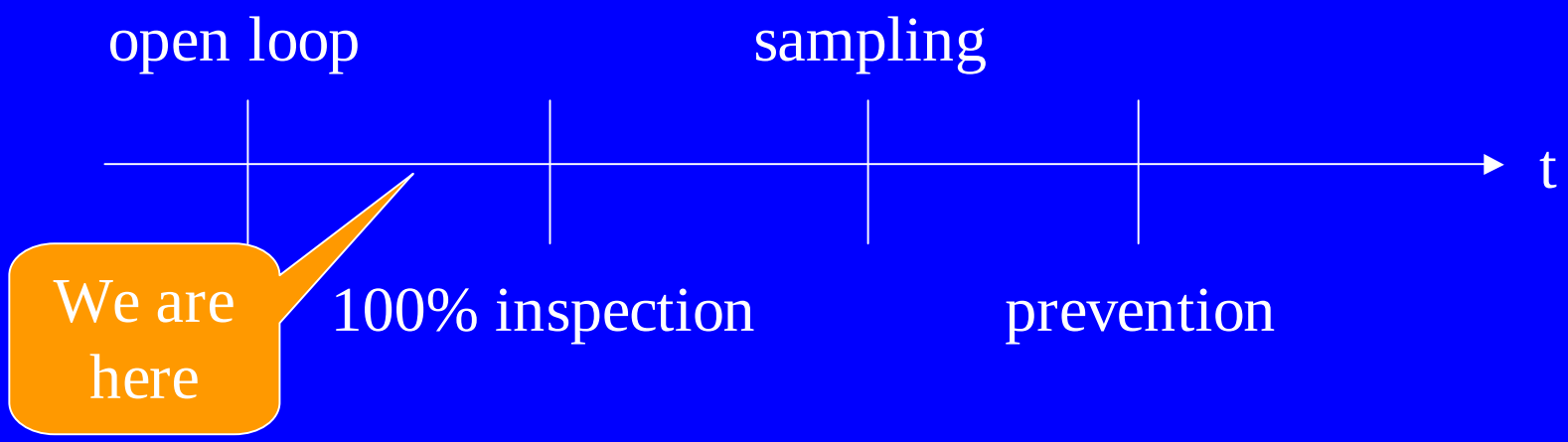
Copyright © 2003, Oxyware Ltd

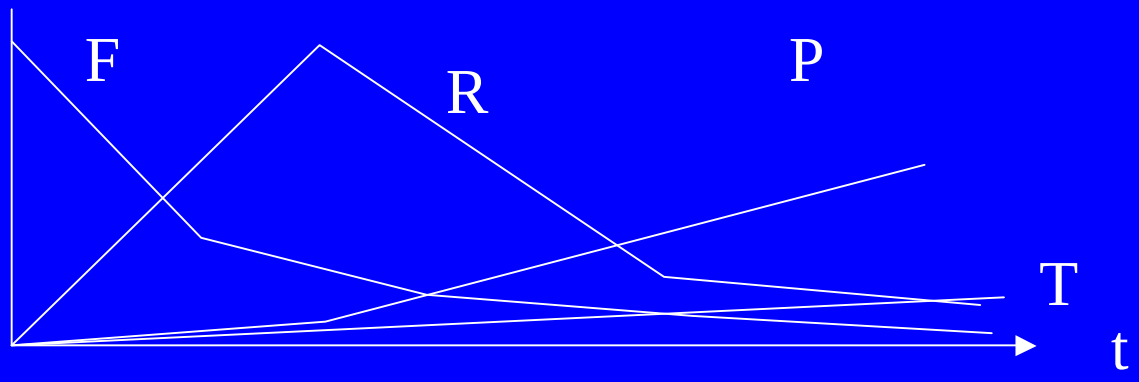
Product v. Process

- Testing gives us feedback about the product
  - Feedback is localised
    - Blame-based ("that code there is wrong!")
    - Knowledge in developers' heads
    - Adaptive learning
- But how does it help us prevent the same errors next time?
  - Need for process feedback loop
    - Contribution-based ("what caused the error?")
    - Knowledge in the world
    - Generative learning
- JFDI v. CMM/ISO 9000 v. XP

Copyright © 2003, Oxyware Ltd

Industry maturity levels

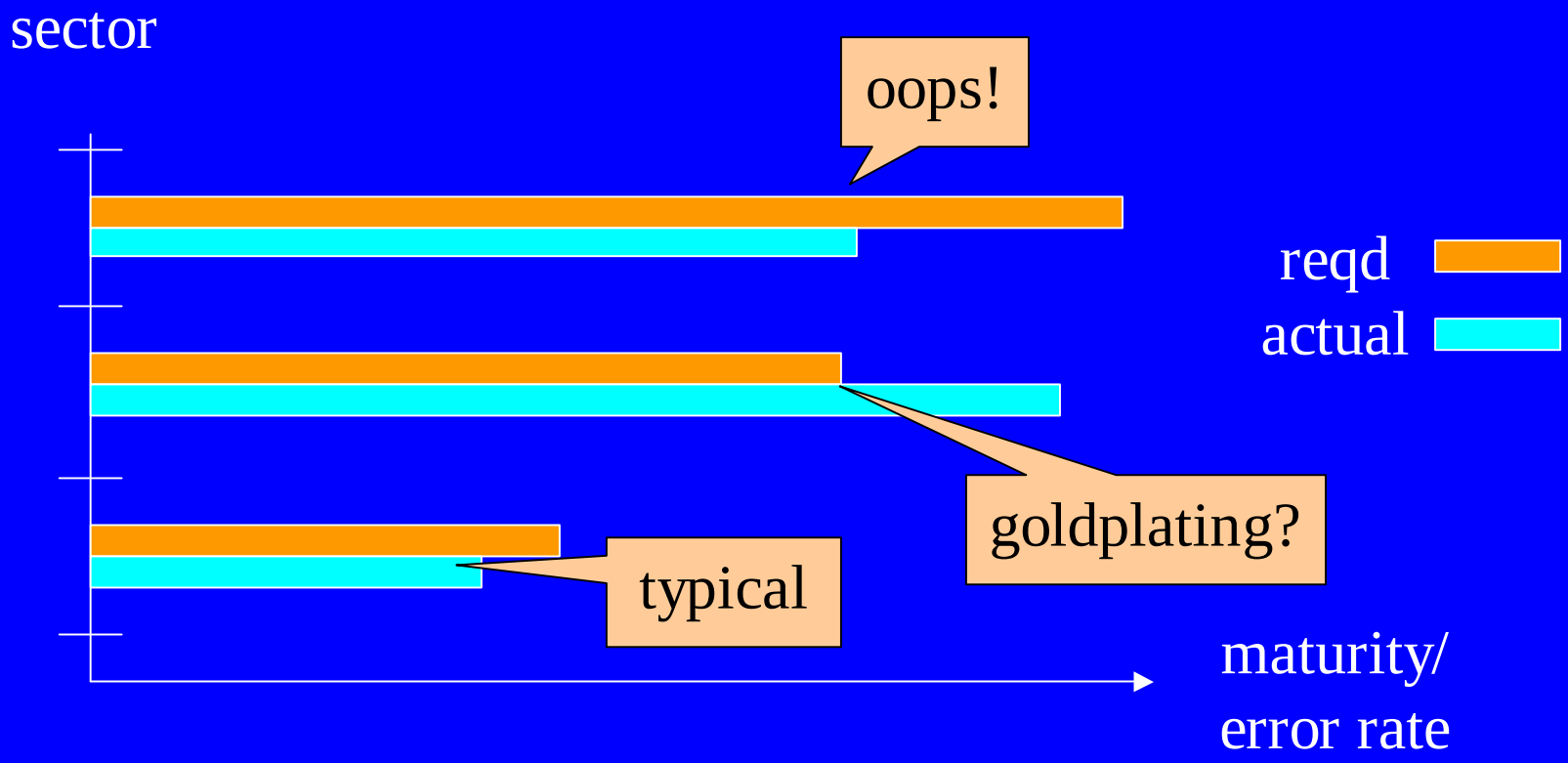




F Failure cost
R Removal cost
P Prevention cost
T Tolerance cost

Copyright © 2003, Oxyware Ltd

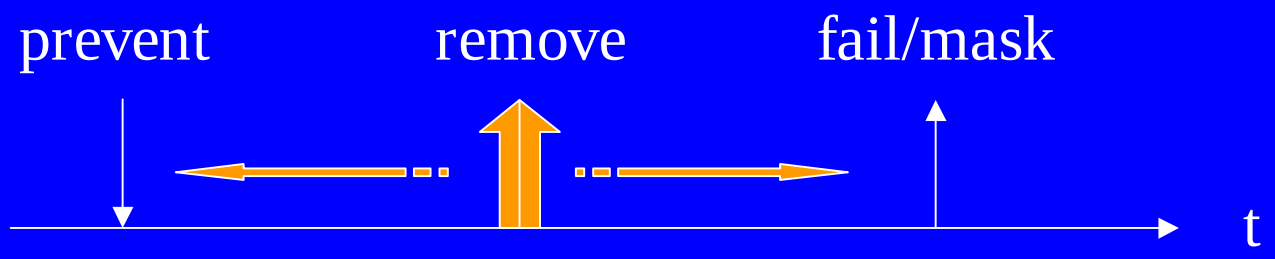
Industry sector tradeoffs



- Trading productivity for quality?
- What is "good enough" for your customers?

Copyright © 2003, Oxyware Ltd

Removal is a mug's game



- With maturity, removal activities will migrate to prevention or tolerance - less testing or reviewing
- Prevention for things that can be avoided
  - Garbage collection, pick lists, syntax-sensitive editors, static allocation, smart pointers
  - 100% prevention, proof-based
- Tolerance for those that can't
  - RAID, Jini, run-time input, exception safety, database transactions, open plane tickets

Copyright © 2003, Oxyware Ltd

Prevention is better than cure

- Over time, more effort is placed earlier in the process
  - Earlier validation of choices
    - Configuration checking, type checking
    - N.B. type checking - removal not prevention
  - Prevention techniques
  - Mistakes can be rectified sooner (= cheaper)
- How can we do earlier checking but with later binding?
  - Rely less on checking!
  - Split interface and implementation - partial bind

Copyright © 2003, Oxyware Ltd

Partial v. complete composition

- Bind early to the intent and late to the mechanism
  - Design by contract, abstract base classes, POP3, getenv
  - Invasive, early choice about allowable combinations
- Compare this with templates or signature-based polymorphism (Python, Smalltalk, C++ STL)
  - All late binding, ad hoc combinations possible
    - Fax machine or modem negotiation of capabilities

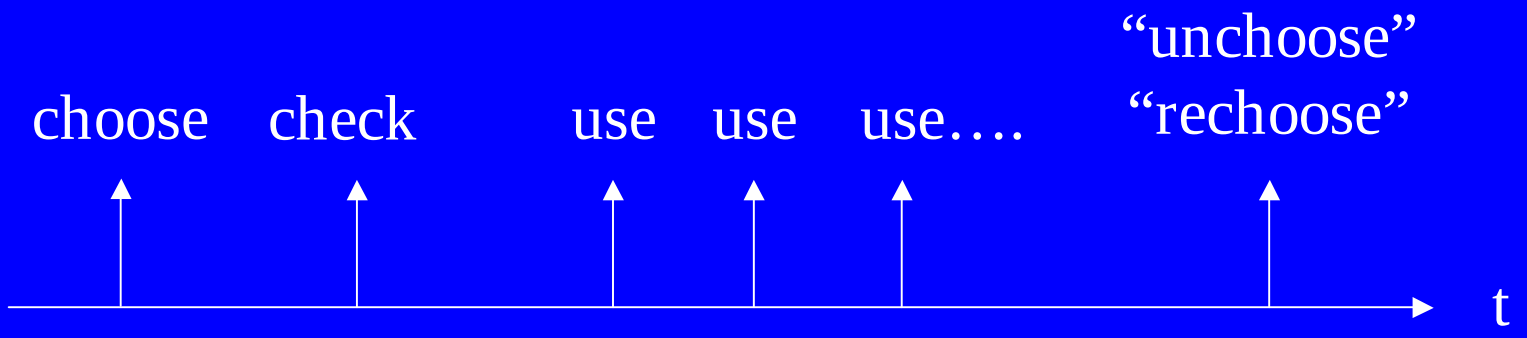
Copyright © 2003, Oxyware Ltd

Binding time adapters

- Early to late - lazy, flexibility-oriented
  - Bind early to i/f and late to implementation
  - proxies, futures
- Late to early - eager, performance-oriented
  - Compiling interpreted languages, code generators
  - Caching, memoisation, read-ahead
  - Optimising compilers - constant lifting, inlining
  - Mostly benefiting from already fixed choices

Copyright © 2003, Oxyware Ltd

Timespan of choice



- "Unchoose" can be end-of-life
  - Destructor, cleanup, unholy mess, vendor lock-in
- Or a "rechoose"
  - Rebind, reselection, refactoring
- Original choice is invariant over the timespan
  - TOCTOU potential
- Timespan of choice is a design decision

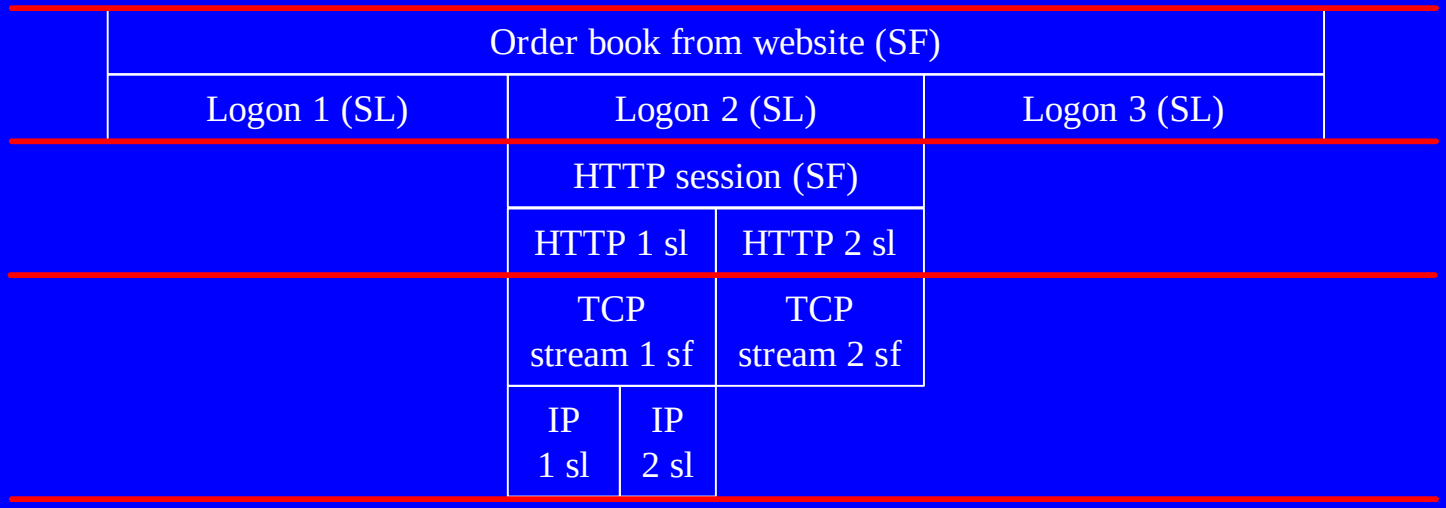
Copyright © 2003, Oxyware Ltd

Concurrency and locking

- Lock granularity = timespan length
  - Coarse-grained locking
    - Certain, slower, longer timespan of choice (table)
  - Fine-grained locking
    - Faster, more concurrency, possible deadlock (row)
- Lock binding time
  - Pessimistic = early bind
    - One case where early bind is slower
  - Optimistic = late bind
    - Metadata (version number), delayed failure on writeback

Copyright © 2003, Oxyware Ltd

Timespan layering



- Higher stateful part of layer extends the timespan of the lower stateless part
- The stateless parts have the same span as the SF part of the layer below

Copyright © 2003, Oxyware Ltd

Conclusion

- The Choose, Check & Use pattern provides a framework and vocabulary for discussing:

The Timing and Cost of Choices

Copyright © 2003, Oxyware Ltd