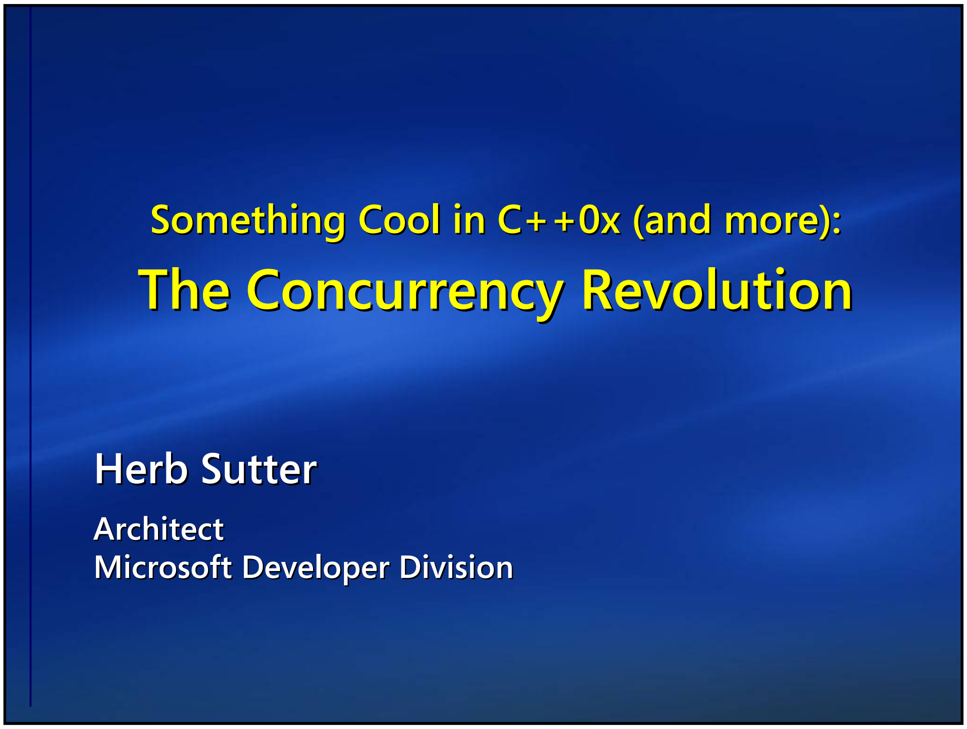
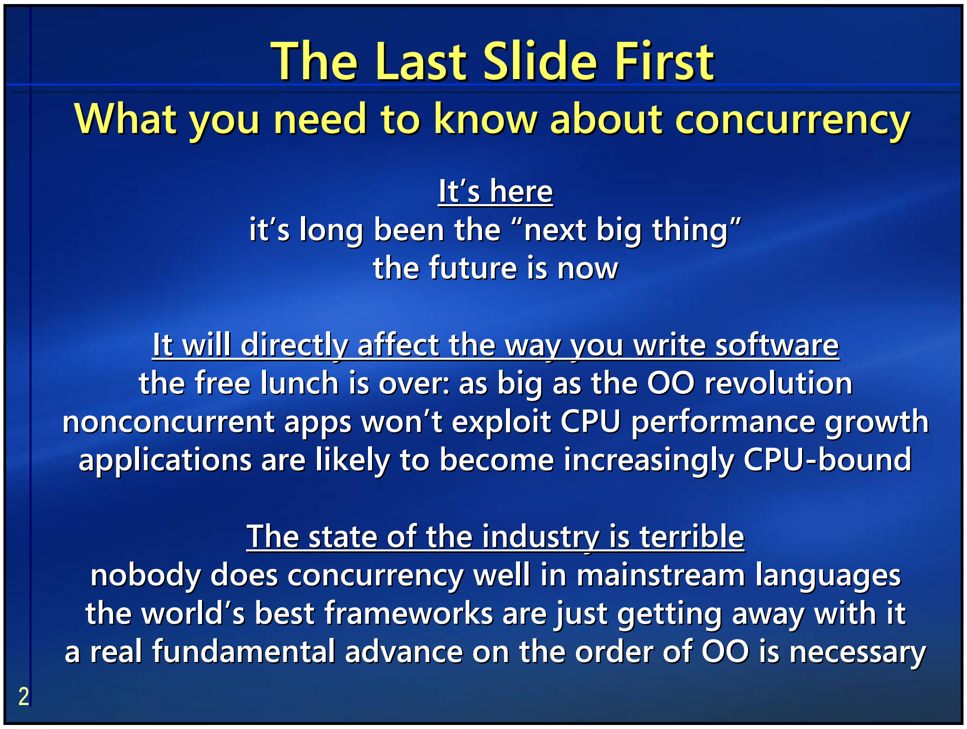


Something Cool in C++0x (and more):

The Concurrency Revolution

Herb Sutter

Architect
Microsoft Developer Division



The Last Slide First

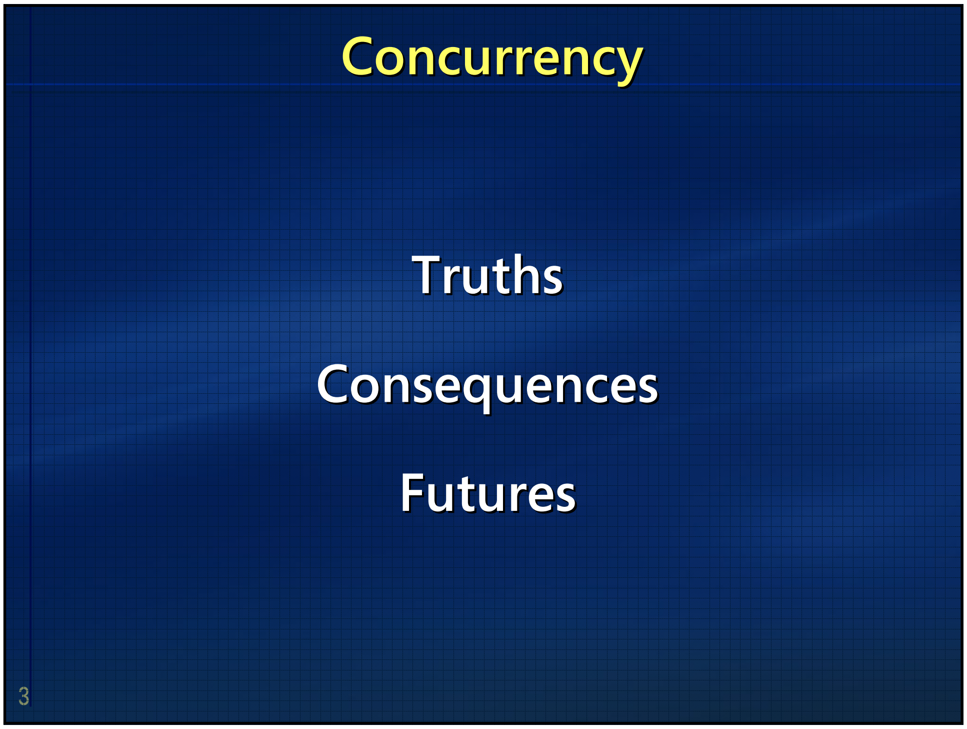
What you need to know about concurrency

It's here
it's long been the "next big thing"
the future is now

It will directly affect the way you write software
the free lunch is over: as big as the OO revolution
nonconcurrent apps won't exploit CPU performance growth
applications are likely to become increasingly CPU-bound

The state of the industry is terrible
nobody does concurrency well in mainstream languages
the world's best frameworks are just getting away with it
a real fundamental advance on the order of OO is necessary

2



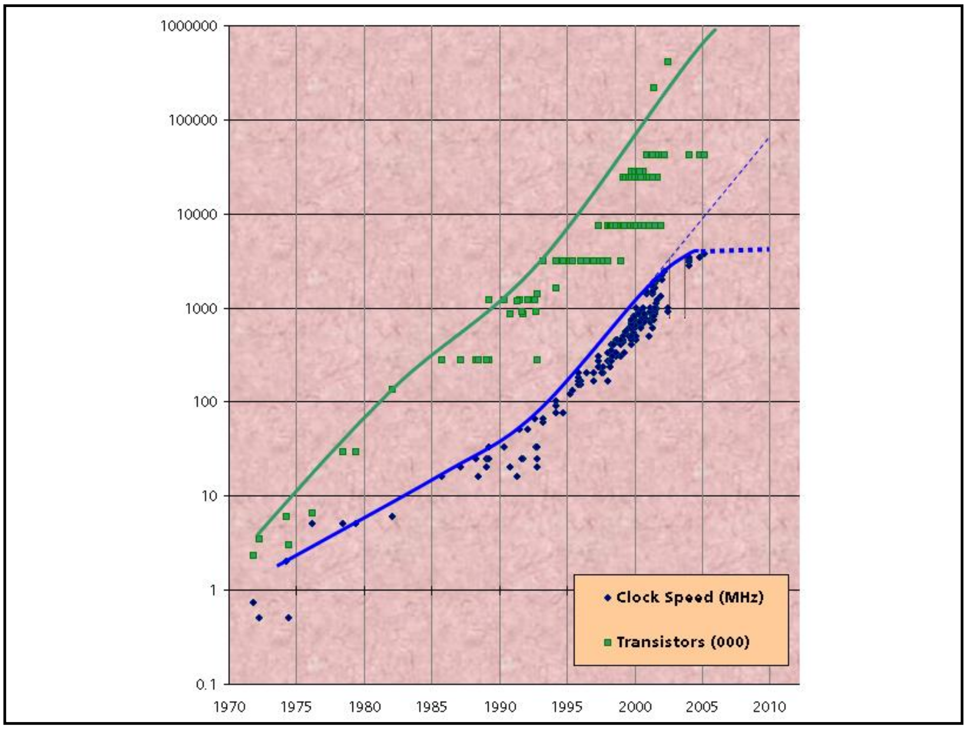
Concurrency

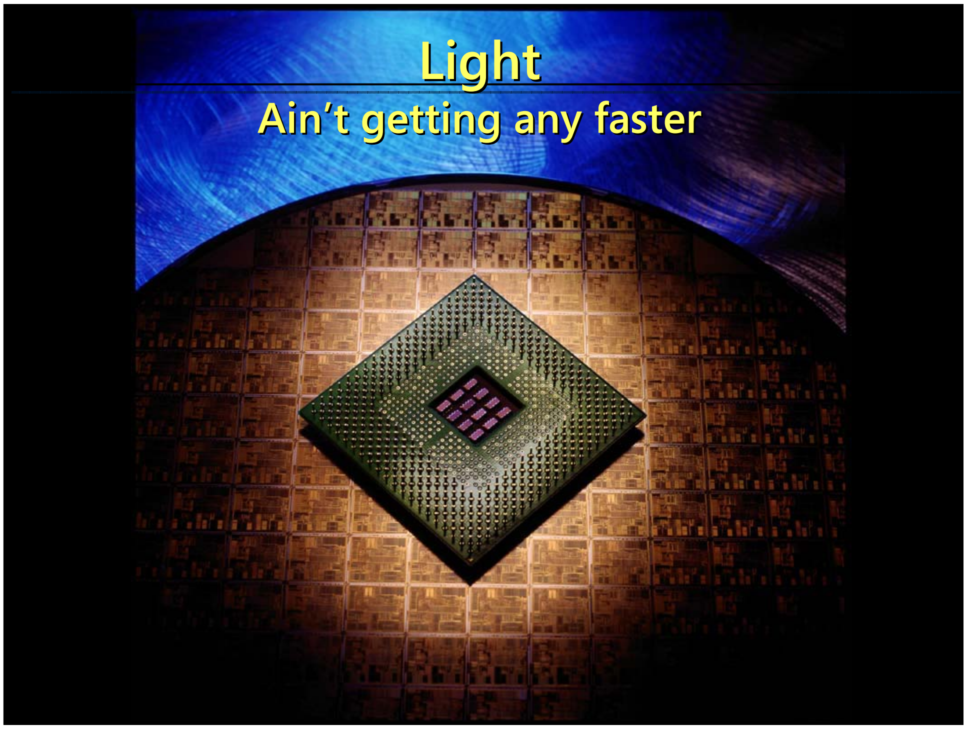
Truths

Consequences

Futures

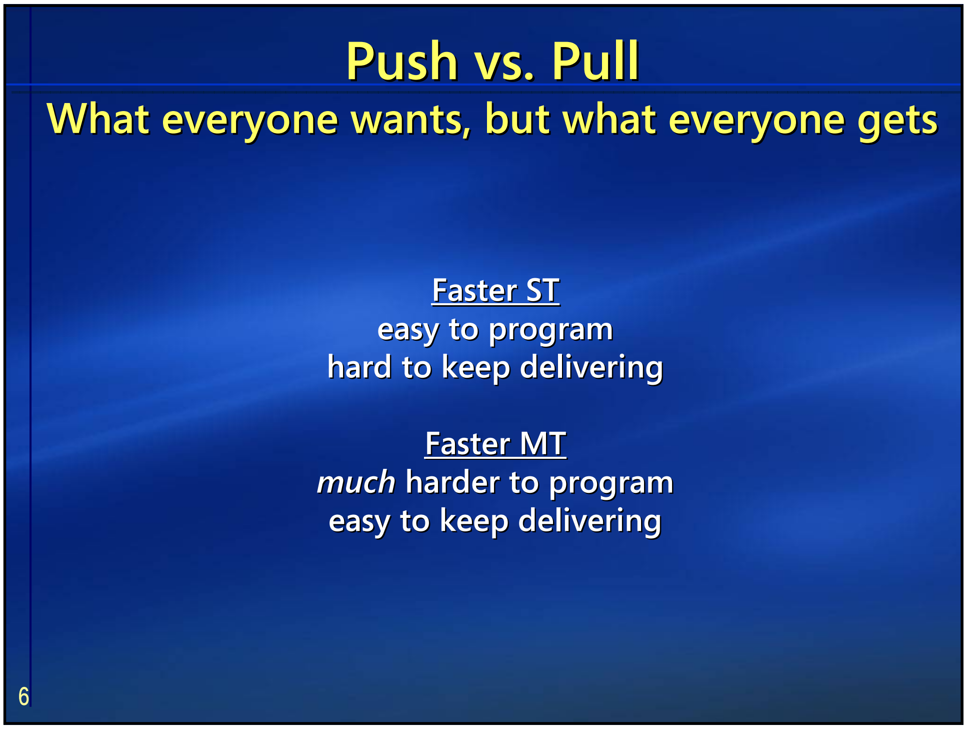
3





Light

Ain’t getting any faster



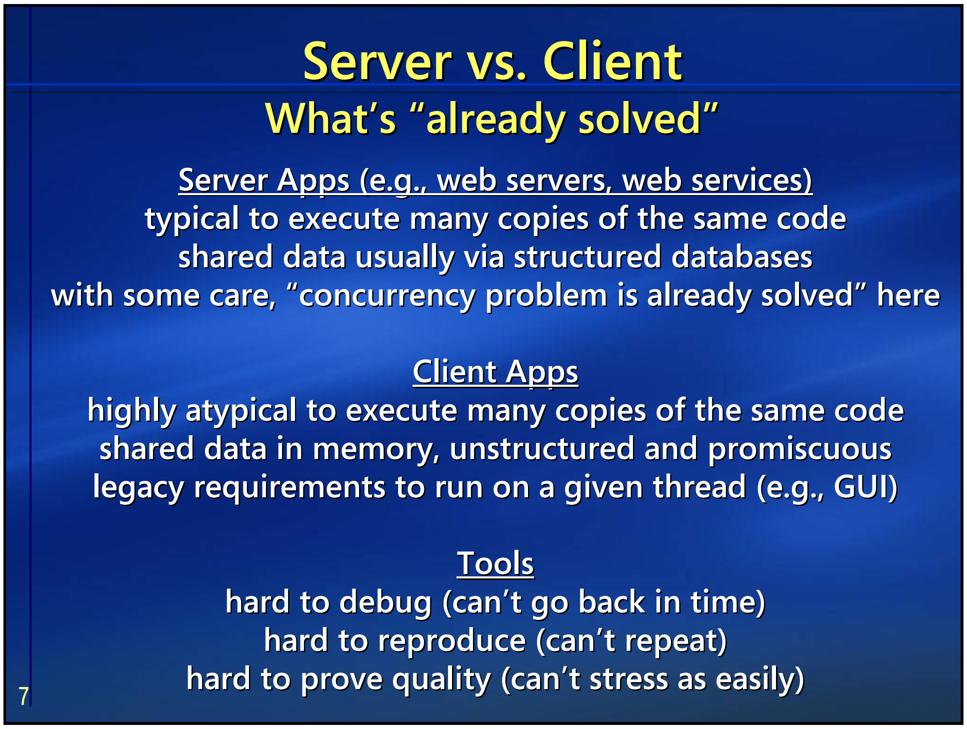
Push vs. Pull

What everyone wants, but what everyone gets

<u>Faster ST</u>
easy to program
hard to keep delivering

<u>Faster MT</u>
much harder to program
easy to keep delivering

6



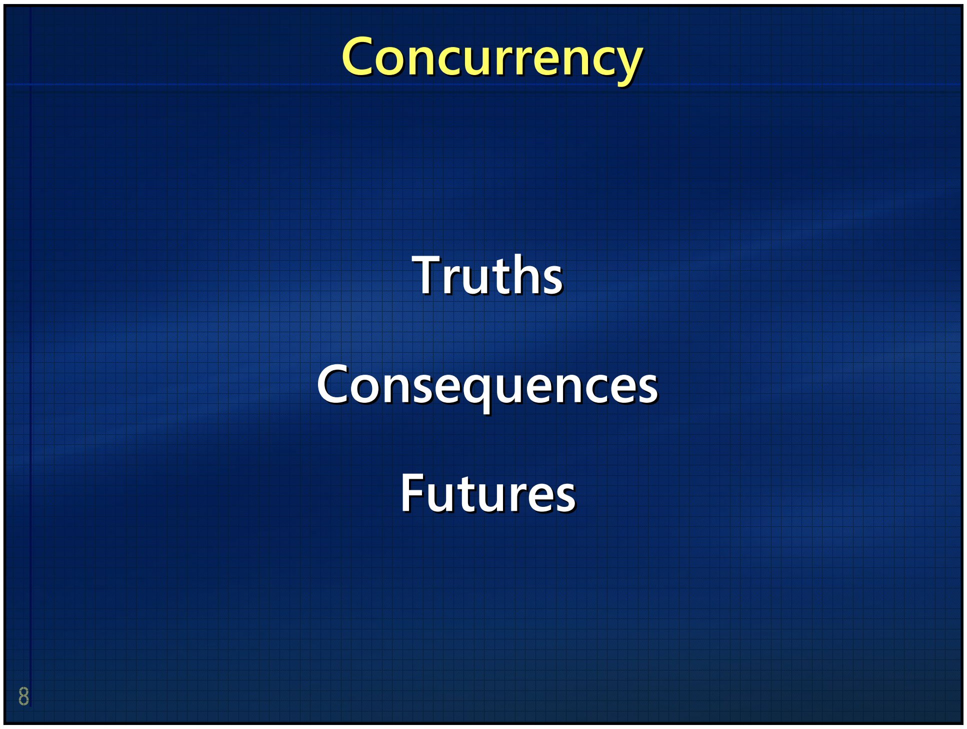
Server vs. Client

What's "already solved"

Server Apps (e.g., web servers, web services)
typical to execute many copies of the same code
shared data usually via structured databases
with some care, "concurrency problem is already solved" here

Client Apps
highly atypical to execute many copies of the same code
shared data in memory, unstructured and promiscuous
legacy requirements to run on a given thread (e.g., GUI)

Tools
hard to debug (can't go back in time)
hard to reproduce (can't repeat)
hard to prove quality (can't stress as easily)

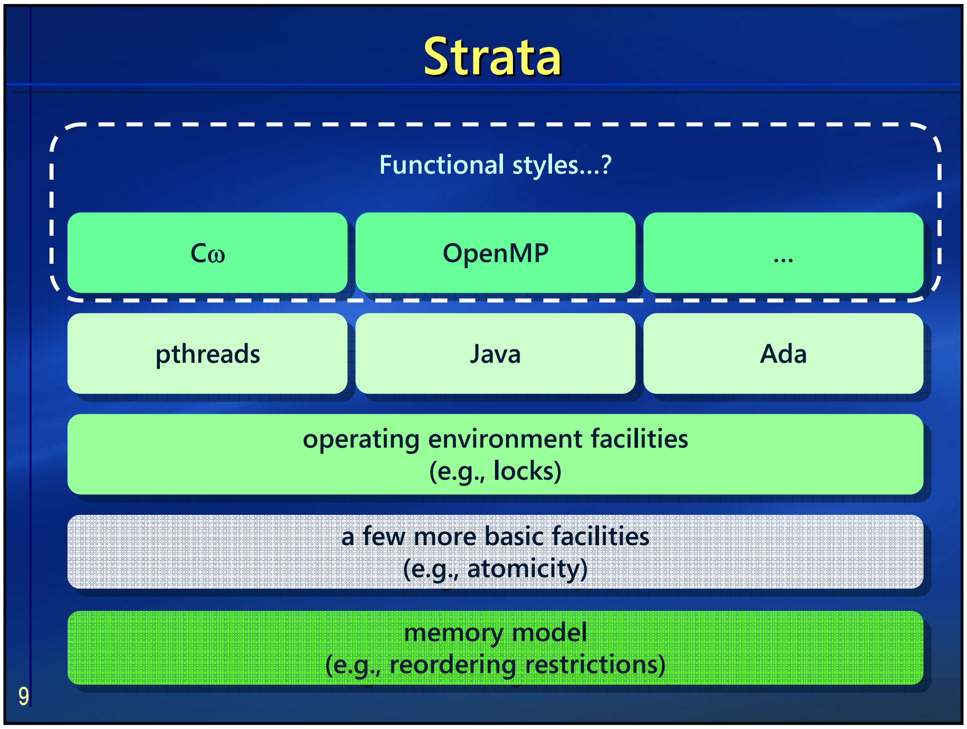


Concurrency

Truths

Consequences

Futures



Strata

Functional styles…?

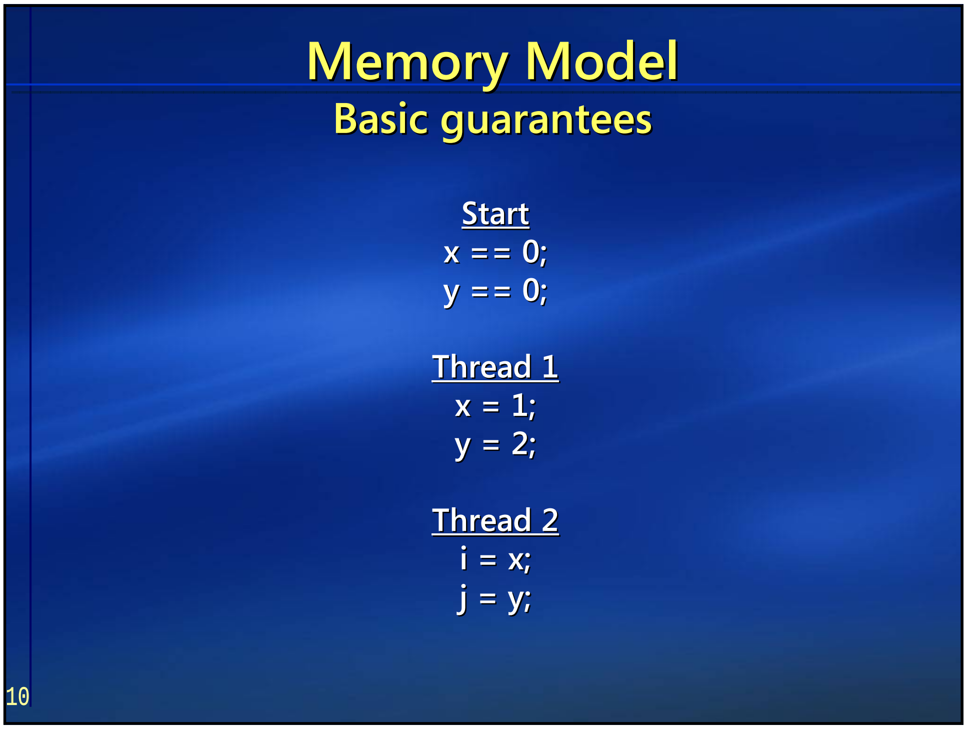
Cω | OpenMP | …

pthreads | Java | Ada

operating environment facilities
(e.g., locks)

a few more basic facilities
(e.g., atomicity)

memory model
(e.g., reordering restrictions)



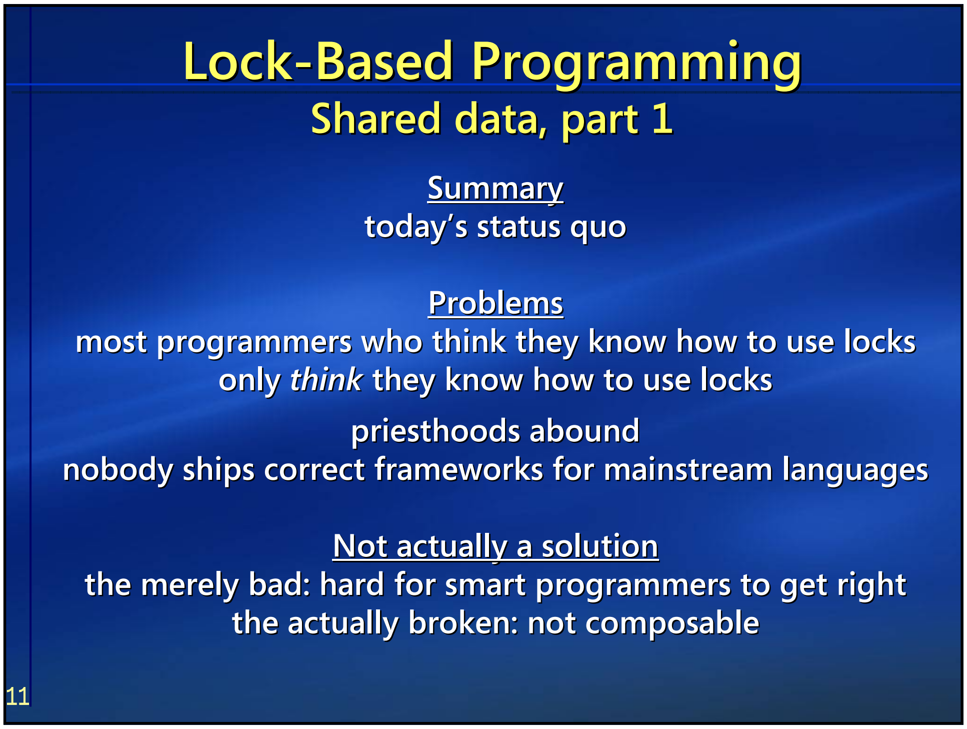
Memory Model

Basic guarantees

Start
x == 0;
y == 0;

Thread 1
x = 1;
y = 2;

Thread 2
i = x;
j = y;



Lock-Based Programming

Shared data, part 1

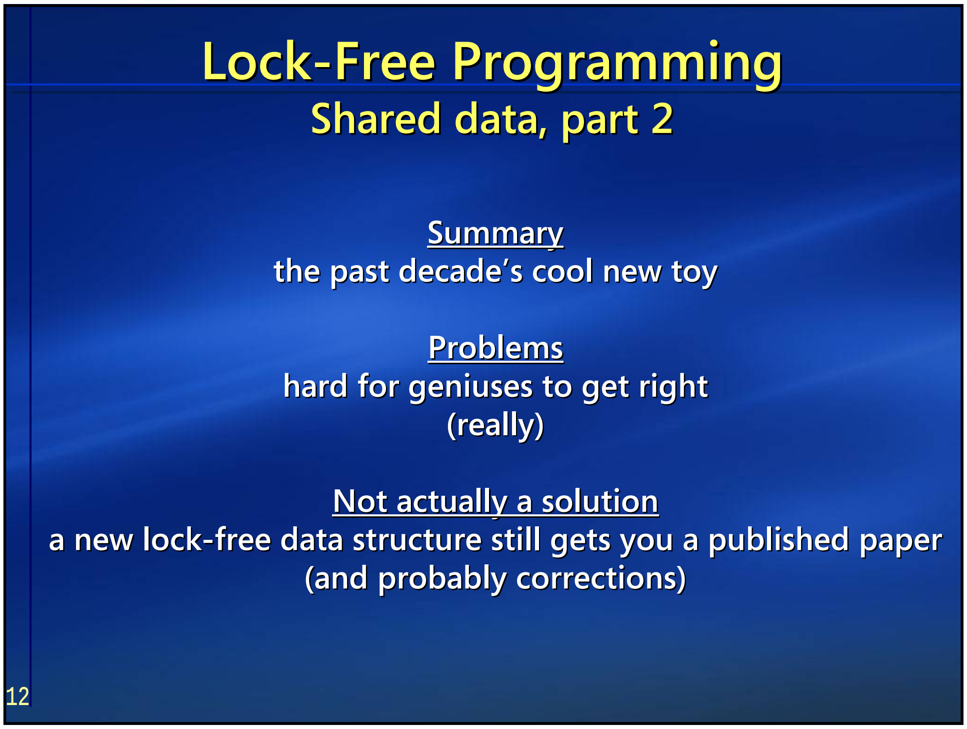
Summary
today’s status quo

Problems
most programmers who think they know how to use locks
only *think* they know how to use locks

priesthoods abound
nobody ships correct frameworks for mainstream languages

Not actually a solution
the merely bad: hard for smart programmers to get right
the actually broken: not composable

11



Lock-Free Programming

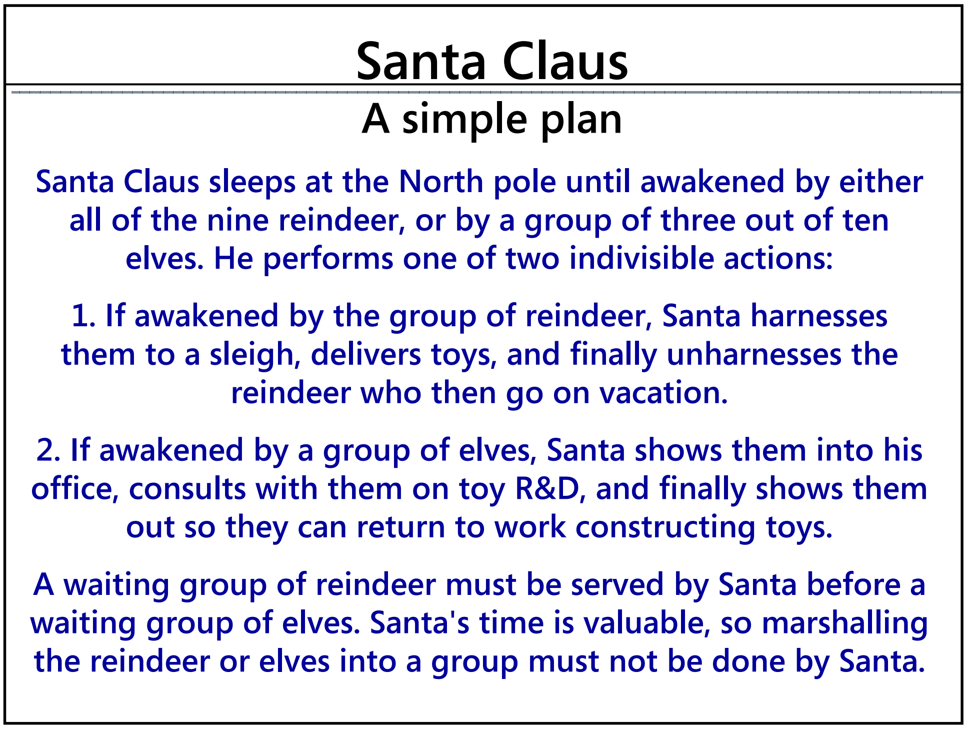
Shared data, part 2

Summary
the past decade’s cool new toy

Problems
hard for geniuses to get right
(really)

Not actually a solution
a new lock-free data structure still gets you a published paper
(and probably corrections)

12



Santa Claus

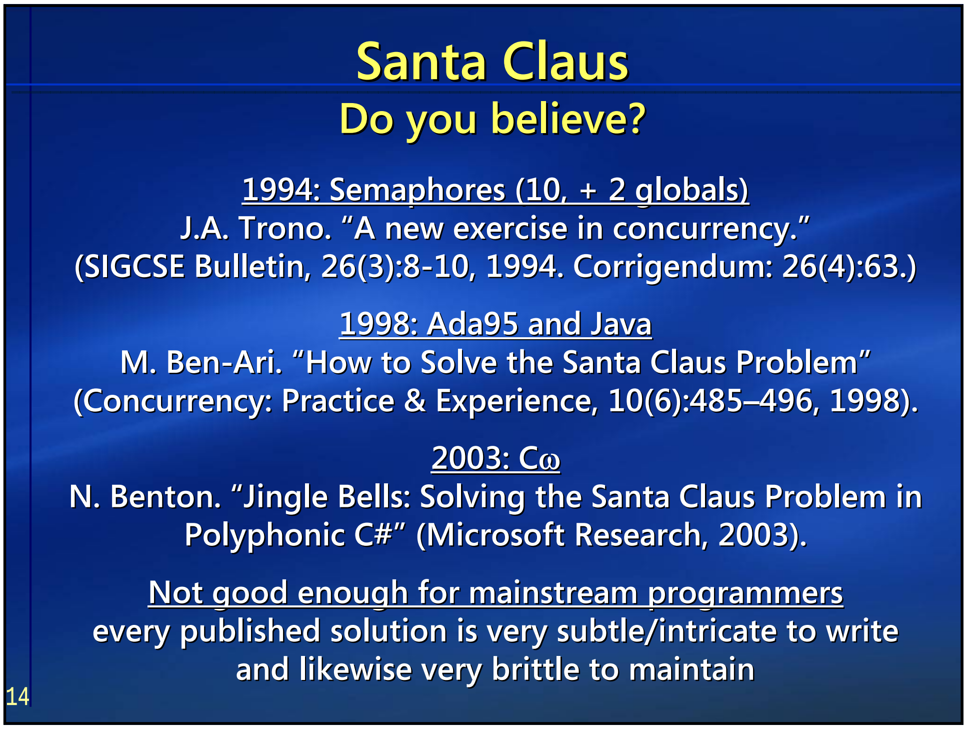
A simple plan

Santa Claus sleeps at the North pole until awakened by either all of the nine reindeer, or by a group of three out of ten elves. He performs one of two indivisible actions:

1. If awakened by the group of reindeer, Santa harnesses them to a sleigh, delivers toys, and finally unharnesses the reindeer who then go on vacation.

2. If awakened by a group of elves, Santa shows them into his office, consults with them on toy R&D, and finally shows them out so they can return to work constructing toys.

A waiting group of reindeer must be served by Santa before a waiting group of elves. Santa's time is valuable, so marshalling the reindeer or elves into a group must not be done by Santa.



Santa Claus

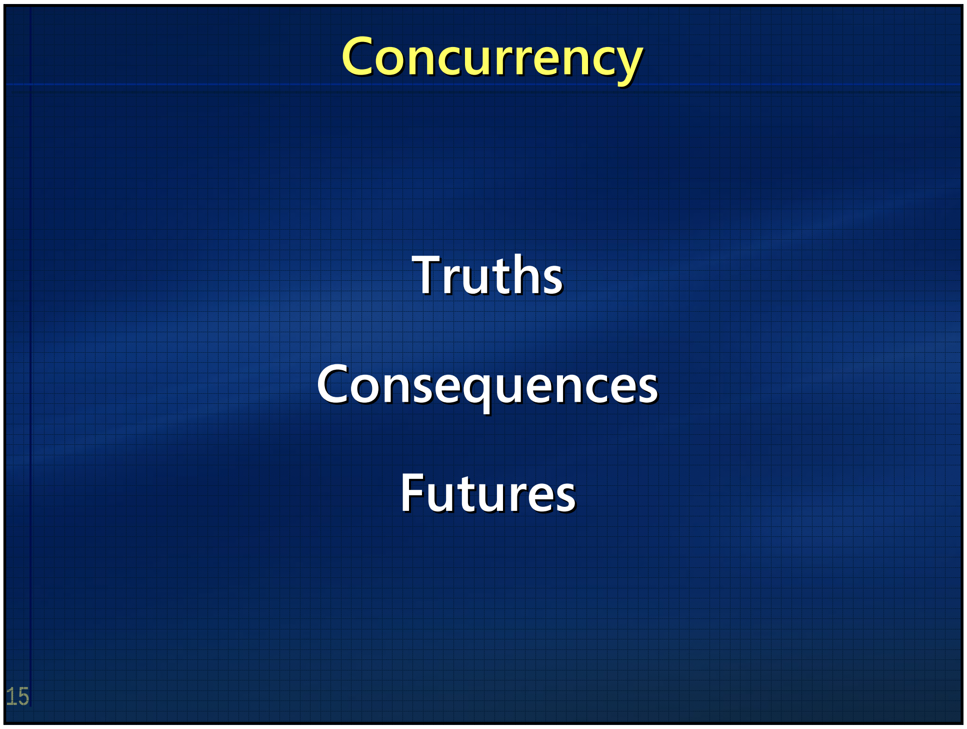
Do you believe?

1994: Semaphores (10, + 2 globals)
J.A. Trono. “A new exercise in concurrency.”
(SIGCSE Bulletin, 26(3):8-10, 1994. Corrigendum: 26(4):63.)

1998: Ada95 and Java
M. Ben-Ari. “How to Solve the Santa Claus Problem”
(Concurrency: Practice & Experience, 10(6):485–496, 1998).

2003: Cω
N. Benton. “Jingle Bells: Solving the Santa Claus Problem in Polyphonic C#” (Microsoft Research, 2003).

Not good enough for mainstream programmers
every published solution is very subtle/intricate to write and likewise very brittle to maintain

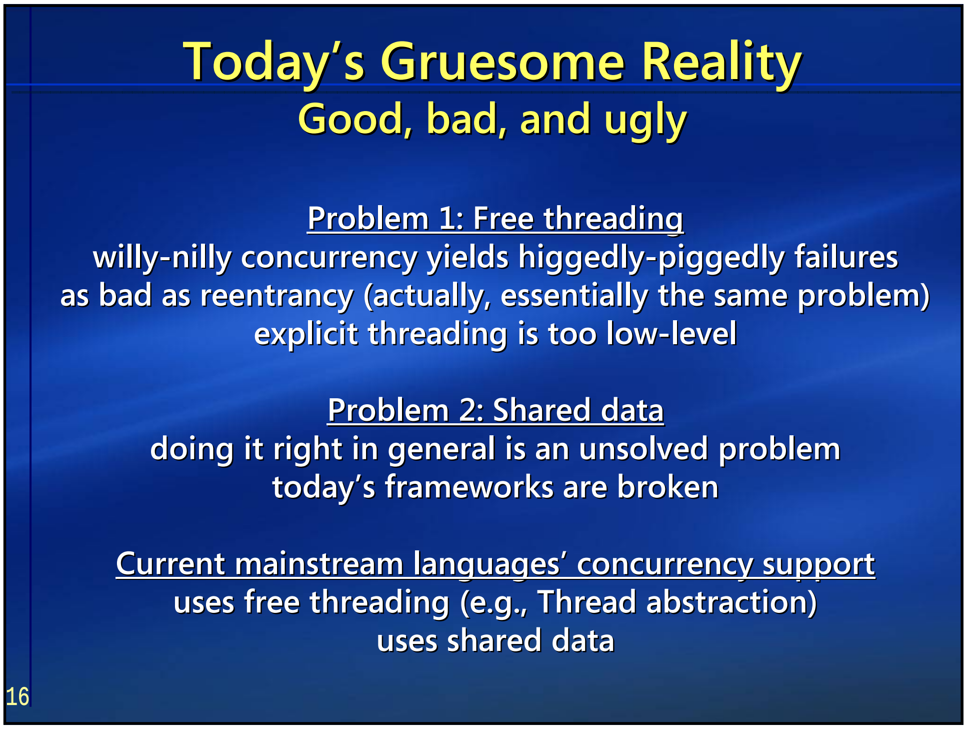


Concurrency

Truths

Consequences

Futures



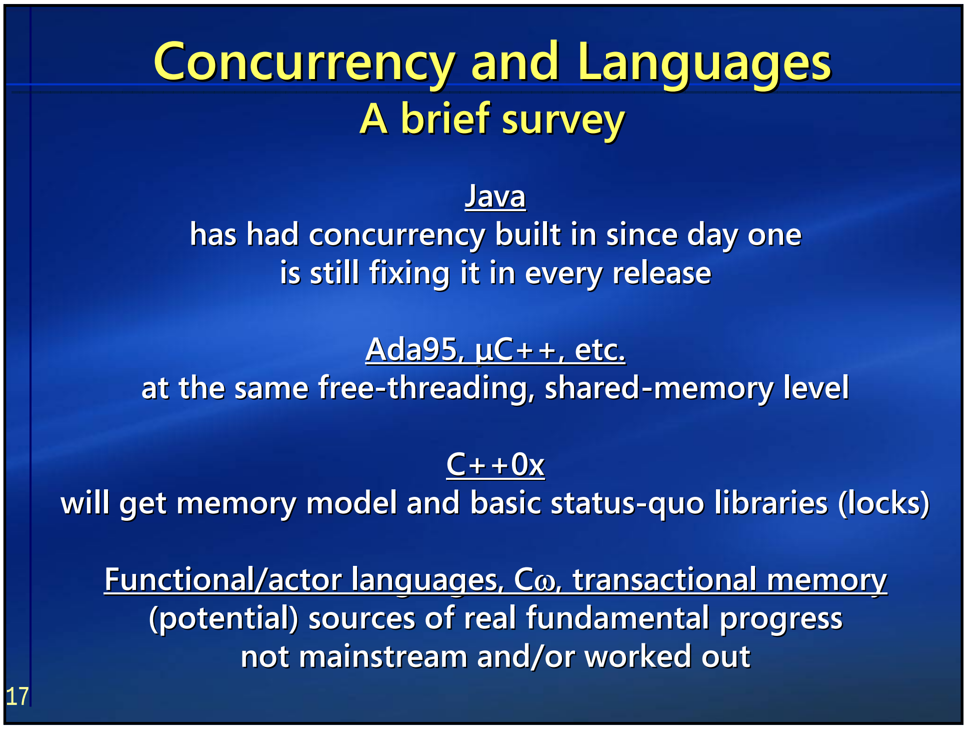
Today's Gruesome Reality

Good, bad, and ugly

Problem 1: Free threading
willy-nilly concurrency yields higgedly-piggedly failures
as bad as reentrancy (actually, essentially the same problem)
explicit threading is too low-level

Problem 2: Shared data
doing it right in general is an unsolved problem
today's frameworks are broken

Current mainstream languages' concurrency support
uses free threading (e.g., Thread abstraction)
uses shared data



Concurrency and Languages

A brief survey

Java

has had concurrency built in since day one
is still fixing it in every release

Ada95, μC++, etc.

at the same free-threading, shared-memory level

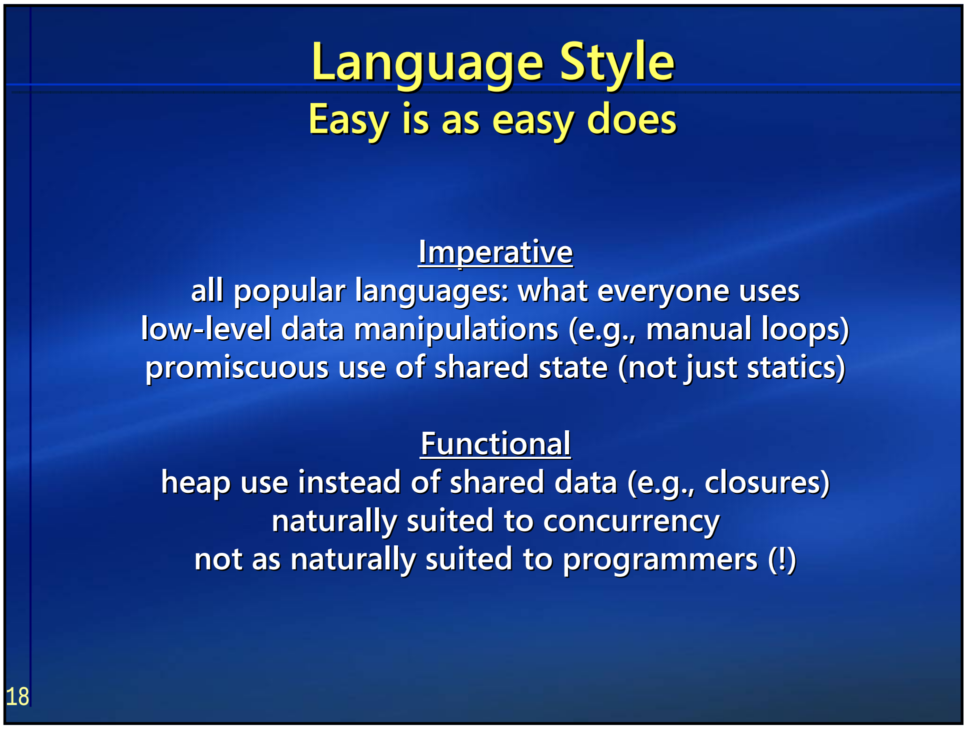
C++0x

will get memory model and basic status-quo libraries (locks)

Functional/actor languages, Cω, transactional memory

(potential) sources of real fundamental progress
not mainstream and/or worked out

17



Language Style

Easy is as easy does

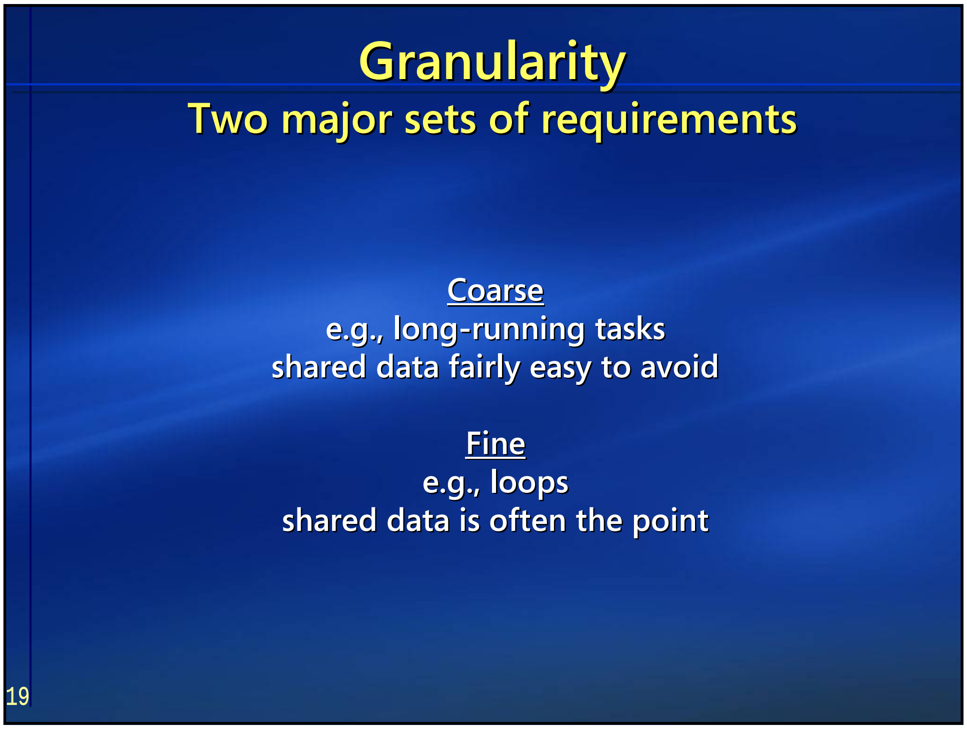
Imperative

all popular languages: what everyone uses
low-level data manipulations (e.g., manual loops)
promiscuous use of shared state (not just statics)

Functional

heap use instead of shared data (e.g., closures)
naturally suited to concurrency
not as naturally suited to programmers (!)

18



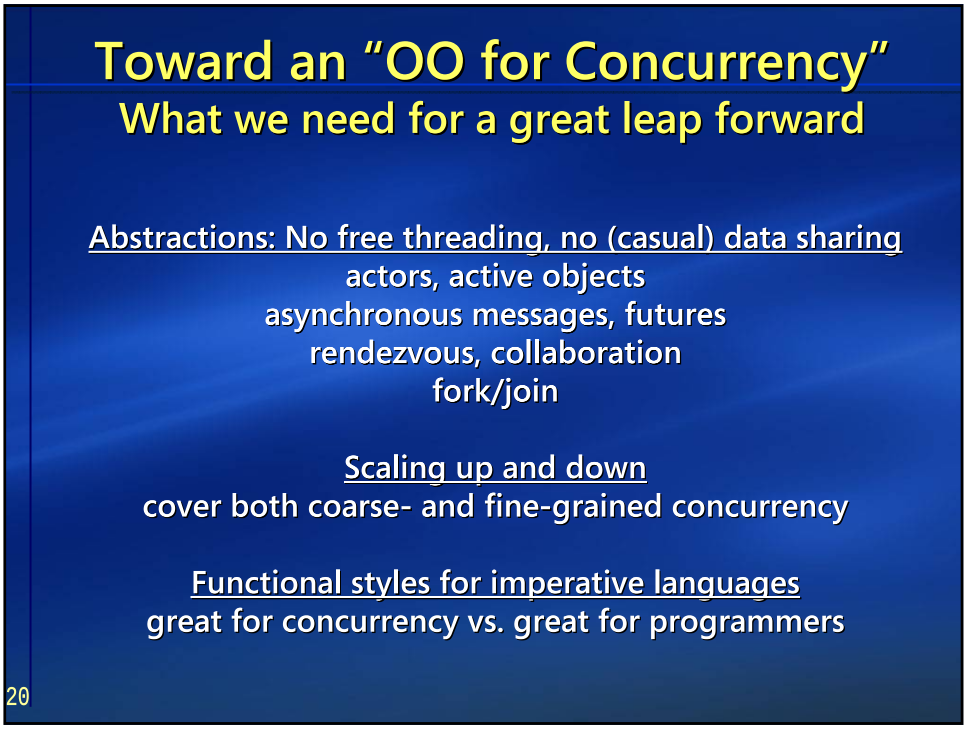
Granularity

Two major sets of requirements

Coarse
e.g., long-running tasks
shared data fairly easy to avoid

Fine
e.g., loops
shared data is often the point

19



Toward an “OO for Concurrency”

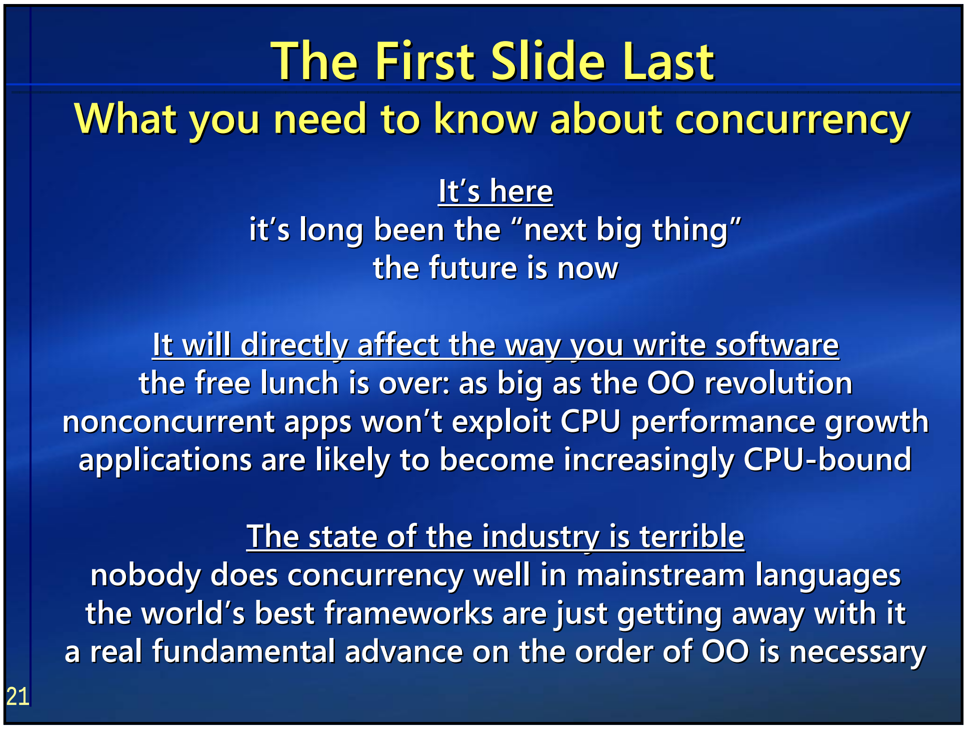
What we need for a great leap forward

Abstractions: No free threading, no (casual) data sharing
actors, active objects
asynchronous messages, futures
rendezvous, collaboration
fork/join

Scaling up and down
cover both coarse- and fine-grained concurrency

Functional styles for imperative languages
great for concurrency vs. great for programmers

20



The First Slide Last

What you need to know about concurrency

It's here

it's long been the "next big thing"
the future is now

It will directly affect the way you write software

the free lunch is over: as big as the OO revolution
nonconcurrent apps won't exploit CPU performance growth
applications are likely to become increasingly CPU-bound

The state of the industry is terrible

nobody does concurrency well in mainstream languages
the world's best frameworks are just getting away with it
a real fundamental advance on the order of OO is necessary

21

Questions?