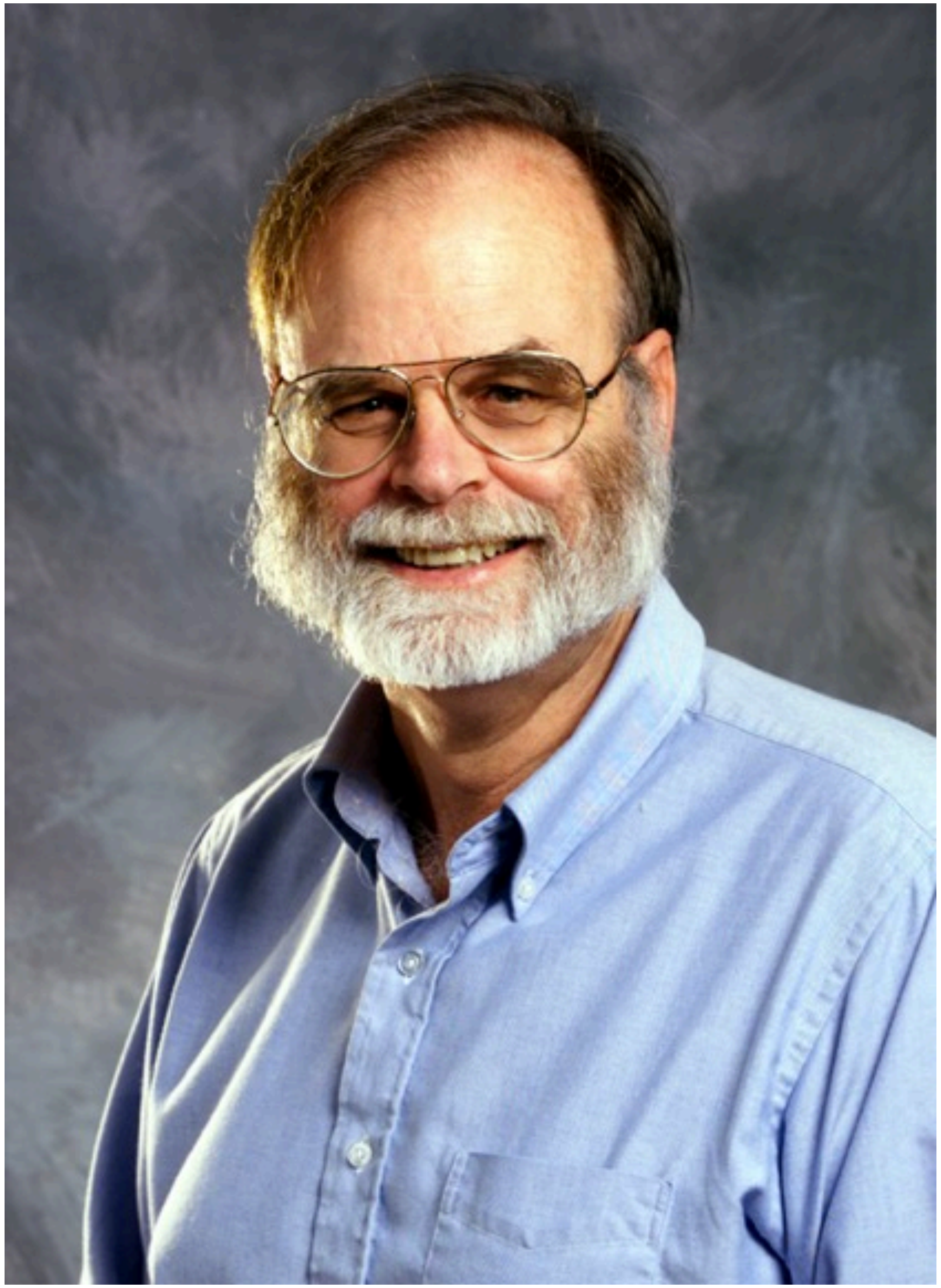
Data Semantics of the Real-time Web

Bryan Cantrill
VP, Engineering

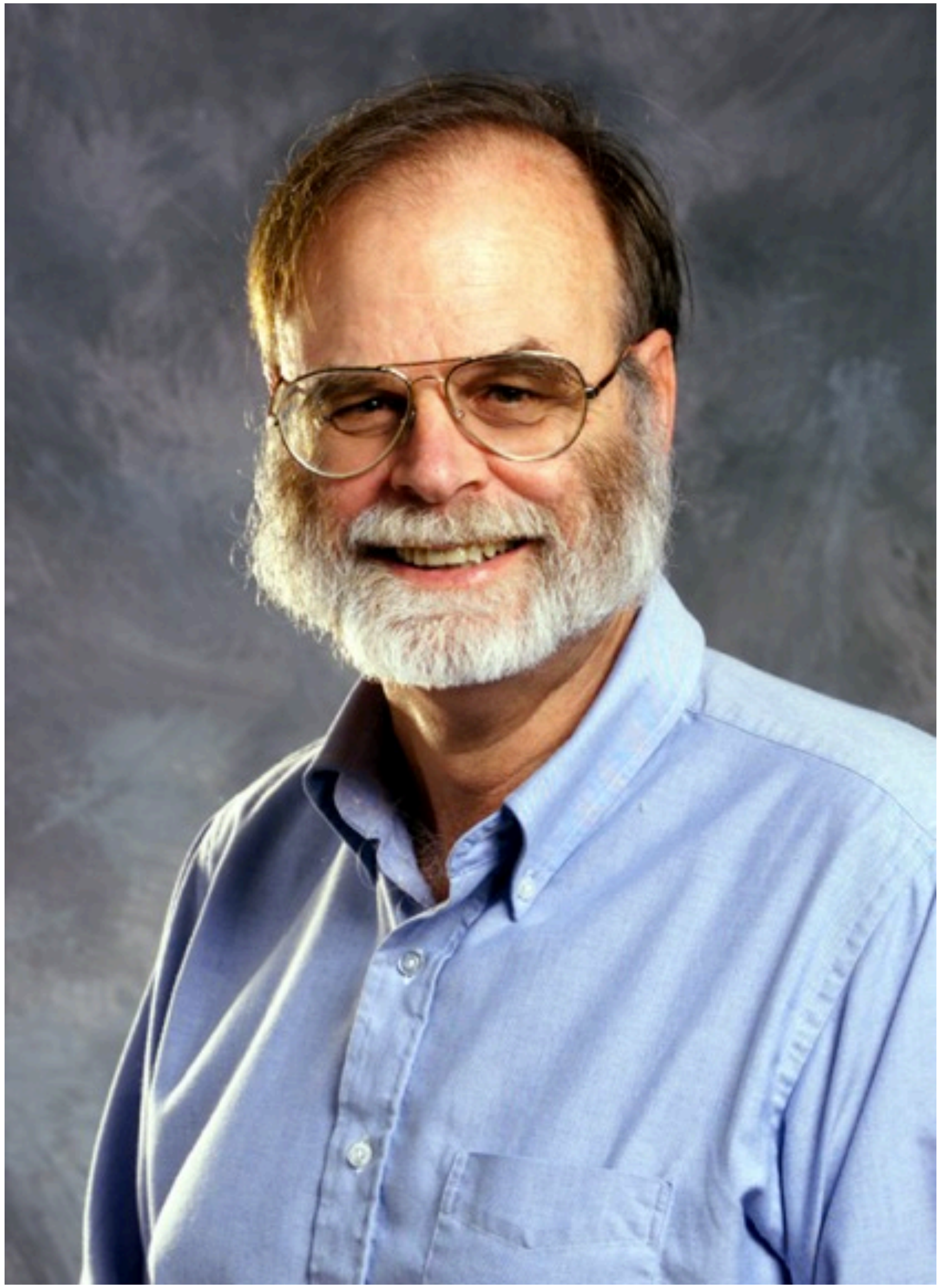
bryan@joyent.com

Data semantics: In the beginning...



Data semantics: In the beginning...

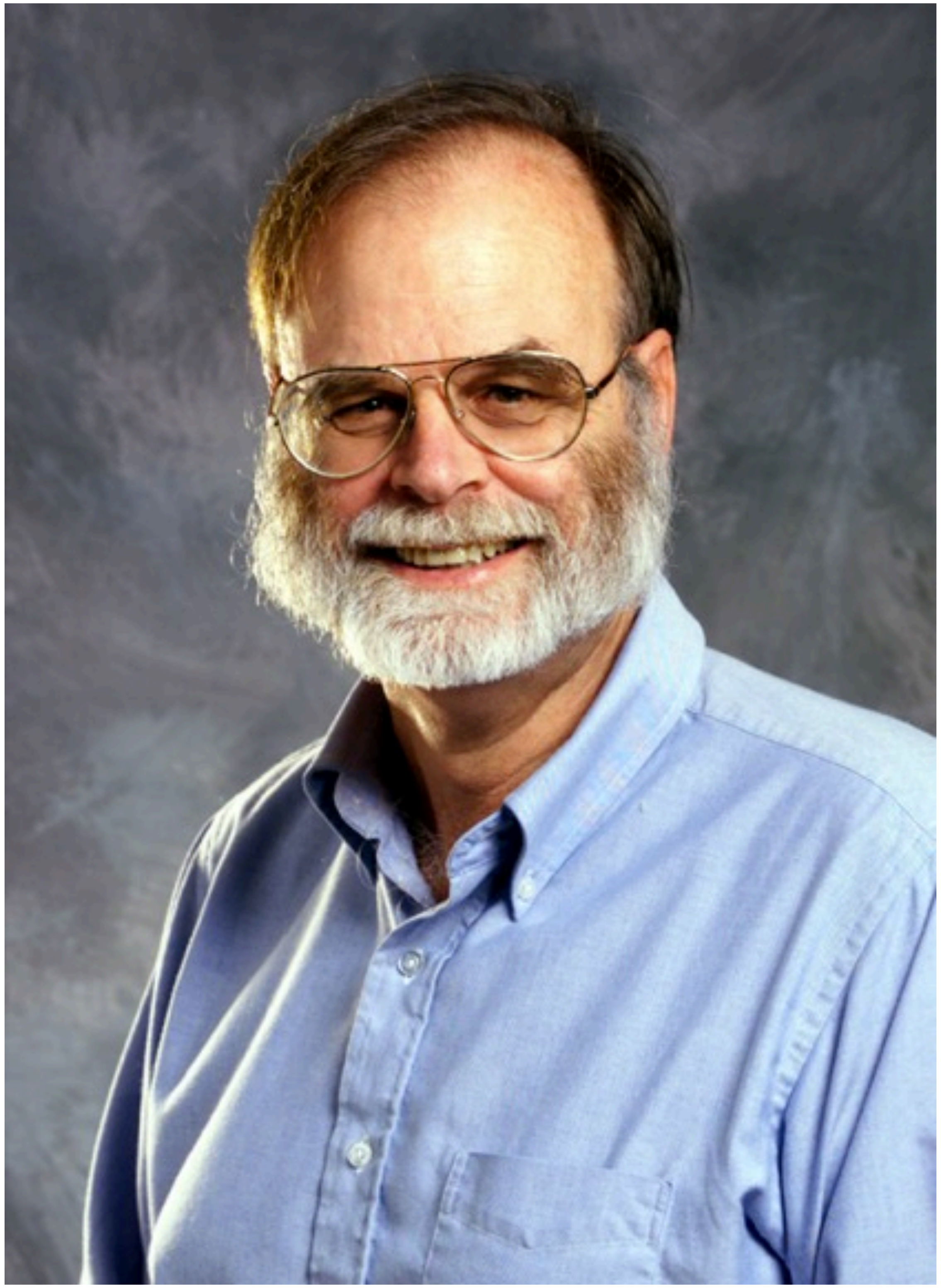
Atomicity



Data semantics: In the beginning...

Atomicity

Consistency

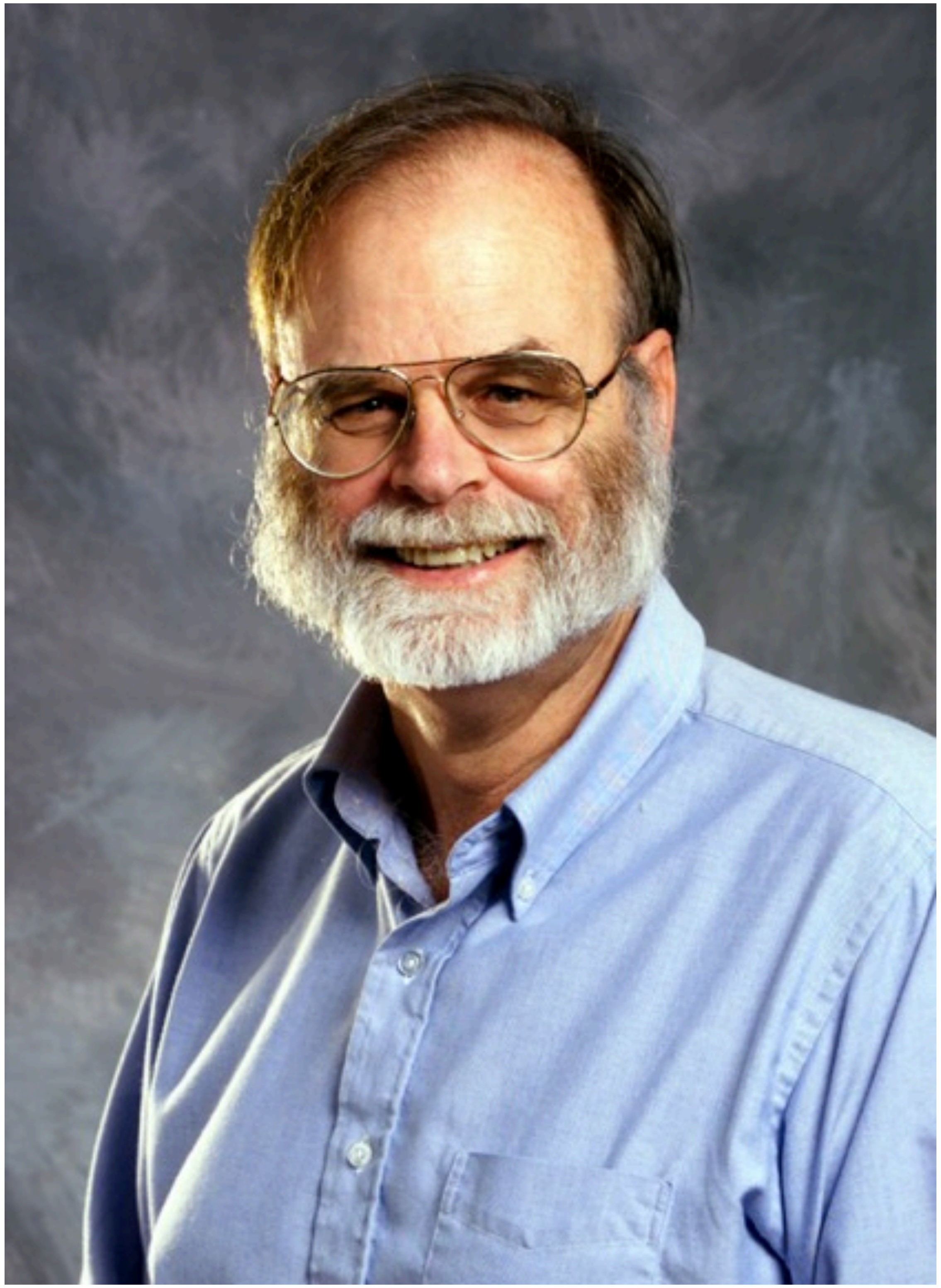


Data semantics: In the beginning...

Atomicity

Consistency

Isolation



Data semantics: In the beginning...

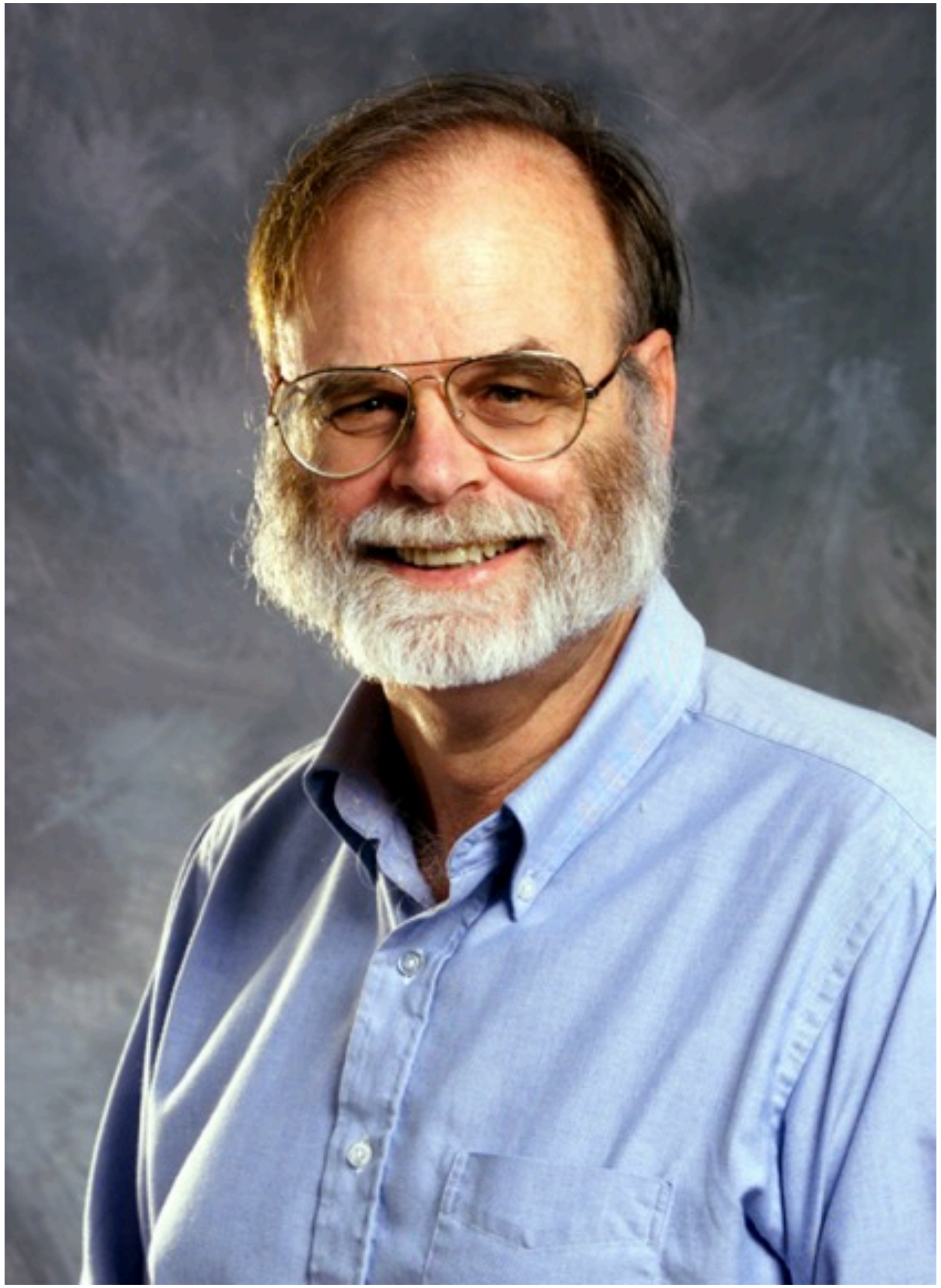
Joyent

Atomicity

Consistency

Isolation

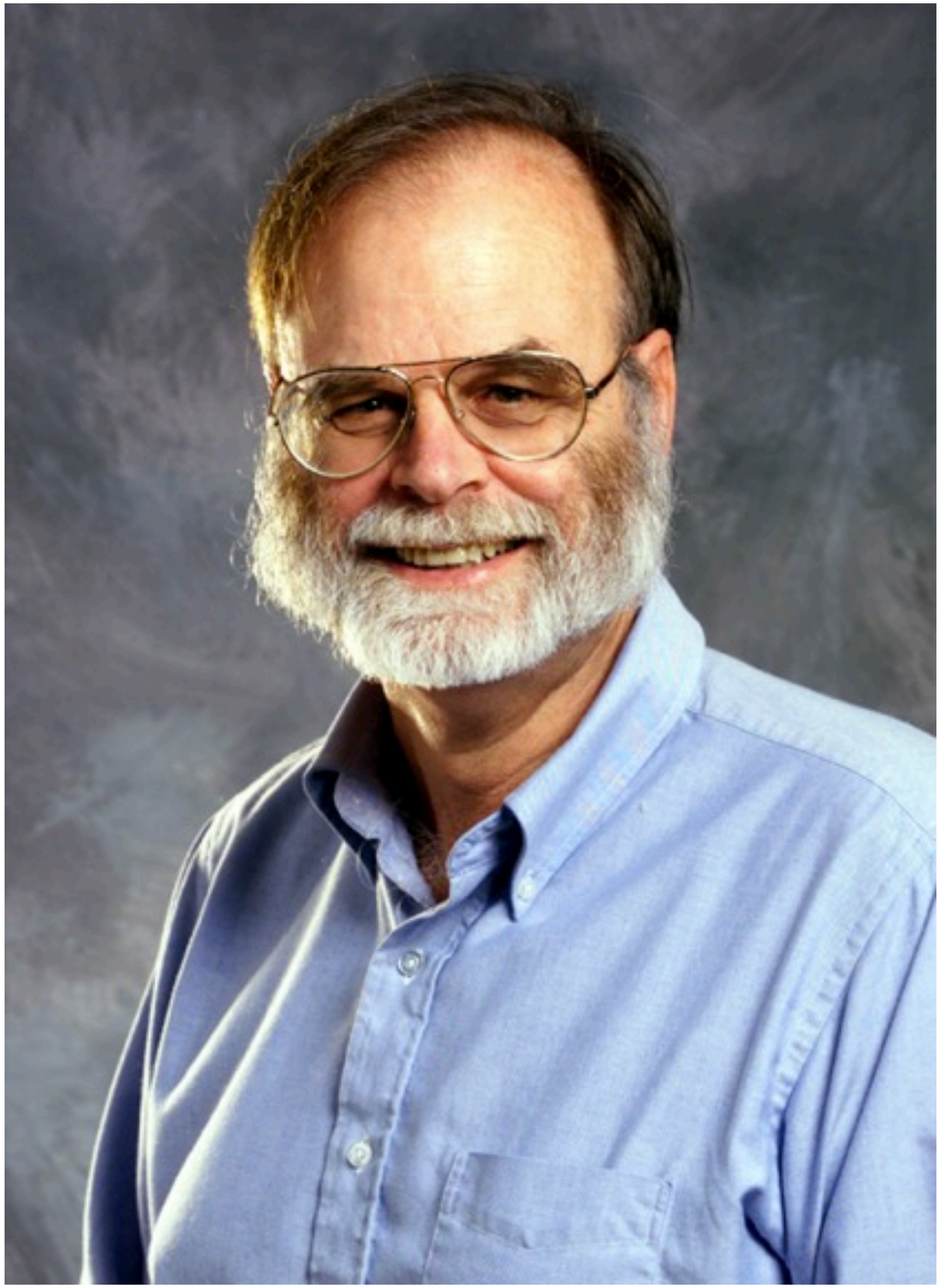
Durability



Data semantics: In the beginning...

Joyent

Atomicity
Consistency
Isolation
Durability



New semantics emerged for scalability

Joyent

New semantics emerged for scalability

Basically
Available

New semantics emerged for scalability

Joyent

Basically
Available
Soft-state

New semantics emerged for scalability

Joyent

Basically
Available
Soft-state
Eventual
 consistency

New semantics emerged for scalability

Joyent

Basically
Available
Soft-state
Eventual
consistency

Changes are afoot again

Joyent

- With its V8 engine, Chrome has initiated a browser war, with JavaScript performance a primary battlefield
- Emerging web sockets standard allows for full duplex, low latency communication from the browser
- Socket.IO provides a JavaScript library that allows web sockets to be adopted without limiting browser support
- Maturity and broad-based adoption of `<canvas>` allows for much more sophisticated visualization in the browser
- The browser has become a sufficiently powerful to allow for interesting *real-time* applications

“Real-time web?”

- Term enjoyed a spike of popularity last year, but there is clearly confusion about the definition of “real-time”
- A *real-time system* is one in which the correctness of the system is relative to its timeliness
- A *hard* real-time system is one which the latency constraints are rigid: violation constitutes total system failure (e.g., an actuator on a physical device)
- A *soft* real-time system is one in which latency constraints are more flexible: violation is undesirable but non-fatal (e.g., a video game or MP3 player)
- The new breed of soft-real time web applications is notable for their real-time *data* semantics

Real-time data semantics

Joyent

- New real-time web applications have dynamic, shared data with an essential temporality: if it's late, it's wrong
- BASE's "eventual consistency" becomes an oxymoron
- Defining "consistency" to be in terms of latency has many ramifications for system design
- But first, a more pressing problem...

Nomenclature!

Nomenclature!

Real-Time

Nomenclature!

Data-Intensive Real-Time

Nomenclature!

Data-Intensive

Real-Time

Nomenclature!

Data-Intensive Real-Time

Nomenclature!

Data-
 Intensive
Real-
Time

Nomenclature!

Data-
Intensive
Real-
Time

Nomenclature!

Data-
Intensive
Real-
Time

A few ramifications of DIRT

Joyent

- System architectures must be rethought with the understanding that the absolute values of latency matter and that there are problems that CDNs can't solve
- Latency bubbles must be accepted as a possibility, and server-side software architectures must be developed to prevent orthogonally cascading latency bubbles
- The stack must be observable not (just) in terms of operations and throughput, but in terms of *latency*

DIRT around the globe

Joyent

- A CDN doesn't bring Singapore any closer to San Diego
- DIRT between continents has constraints imposed by the physics and geography of the problem
- Some constraints will need to be relaxed:
  - Applications may need to be designed assuming worst-case physics/geography (200+ ms)
  - Applications may need to be partitioned into latency domains and executed on geographically distributed compute
- There is unlikely to be one solution that fits all needs; applications will need to become increasingly aware of the latency consequences of geography

Server-side DIRT

Joyent

- Embedded real-time systems are sufficiently controlled that latency bubbles can be architected away
- Web-facing systems are far too sloppy to expect this!
- Focus must shift from preventing latency bubbles to preventing latency bubbles from cascading
- Operations that can induce latency (network, I/O, etc.) must not be able to take the system out with them!
- Implies purely asynchronous and evented architectures, which are notoriously difficult to implement...
- ...but Node.js (a server-side framework based on V8) makes these architectures much more readily attainable

Digging through DIRT

- Where latency implies correctness, we must be able to observe it to verify that the system is operating correctly
- Not good enough to look at traditional scalars (e.g., operations per second); low counts could be due to either light load... or pathological latency
- Must be able to observe the *entire* stack in *production*
- Implies not only dynamic instrumentation (e.g. DTrace) but the tooling to make use of the data, and to phrase ad hoc queries to further instrument
- Visualizing latency is tricky, but there are emerging techniques (e.g., Brendan Gregg's "Visualizing System Latency" in *Communications of the ACM*, July 2010)

Getting DIRT under the fingernails

Joyent

- DIRT poses many challenges, but there is a tremendous desire for the value that it can bring to the web
- DIRT is not going to replace BASE; as BASE with ACID, DIRT will coexist with other data semantics in a larger heterogeneous system
- DIRT forces reconsideration many aspects of system design, including at least geography, server-side architecture and operational observability
- This is a nascent area; there are many more questions than answers!
- Bad news: if it’s late, it’s wrong
- Good news: if it’s old, it’s trash

Let’s get DIRTy!