In-Memory Computing SUMMIT | VIRTUAL EVENT 2020

The Power of In-Memory Computing: From Supercomputing to Stream Processing

William Bain, Founder & CEO
ScaleOut Software, Inc.
October 28, 2020

About the Speaker

Dr. William Bain, Founder & CEO of ScaleOut Software:

- Email: wbain@scaleoutsoftware.com
- Ph.D. in Electrical Engineering (Rice University, 1978)
- Career focused on parallel computing – Bell Labs, Intel, Microsoft

ScaleOut Software develops and markets In-Memory Data Grids, software for:

- Scaling application performance with in-memory data storage
- Providing operational intelligence on live data with in-memory computing
- 15+ years in the market; 450+ customers, 12,000+ servers

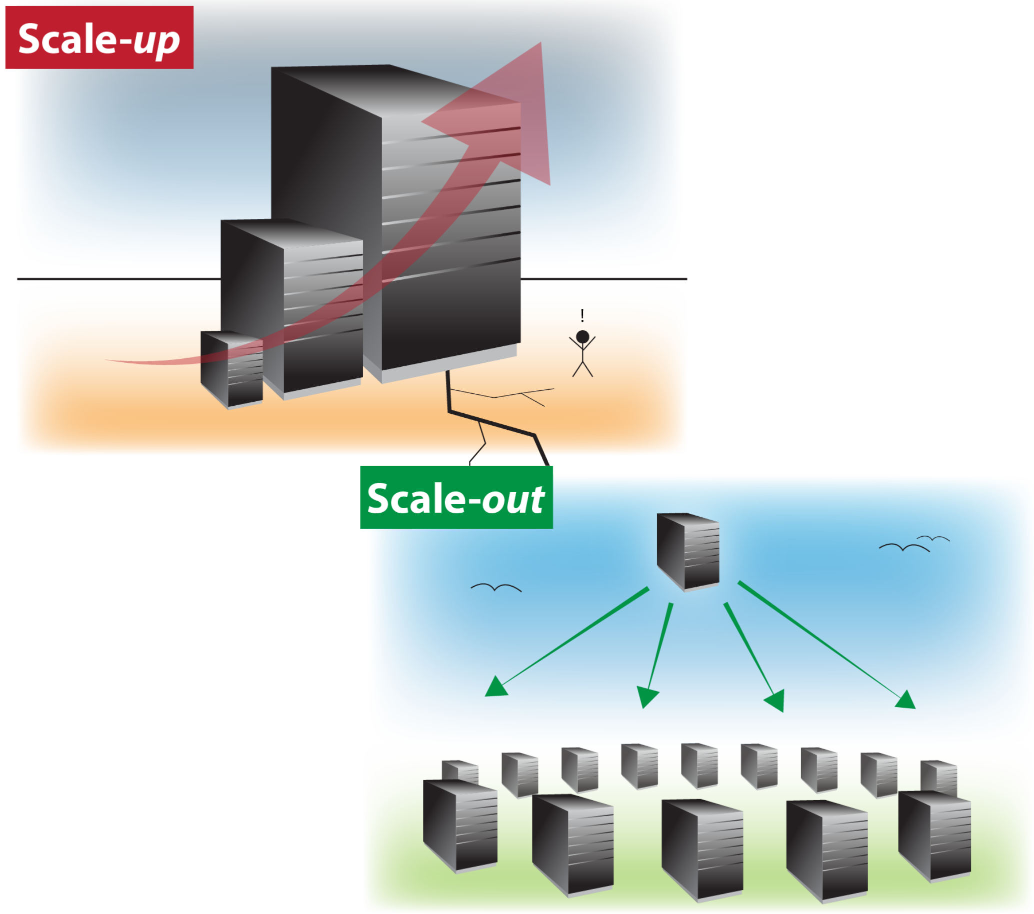
What Is In-Memory Computing?

Generally accepted characteristics:

- Comprises both hardware & software techniques.
- Hosts data sets in primary memory.
- Distributes computing across many servers.
- Employs data-parallel computations.

Why use IMC?

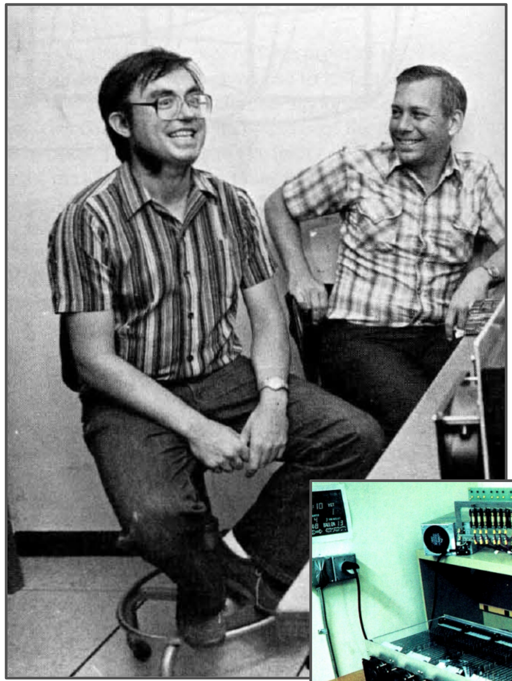
- Can quickly process “live,” fast-changing data.
- Can analyze large data sets.
- “Scaling out” is more scalable and cost-effective than “scaling up”.



In the Beginning

Caltech Cosmic Cube (1983)

- Possibly the earliest in-memory computing system
- Created by professors Geoffrey Fox and Charles Seitz
- Targeted at solving scientific problems (high energy physics, astrophysics, chemistry, chip simulation)
- 64 “nodes” with Intel 8086/8087 processors & 8MB total memory, hypercube interconnect, 3.2 MFLOPS
- “One-tenth the power of the Cray 1 but 100X less expensive”



The Era of Commercialization

Commercial Parallel Supercomputers

- 1984: Industry pioneered by Justin Rattner, Intel
- 1985: Intel iPSC1
  - 80286, 128 nodes, 512MB, hypercube
- 1985: Ncube/10
  - Custom, 1024 nodes, 128MB, hypercube
- 1993: IBM SP1
  - RS/6000, 512 nodes, 128GB, proprietary
- 1993: Intel Paragon
  - I860, 4K nodes, 128GB, 2D mesh
  - 300 GFLOPS

Justin Rattner

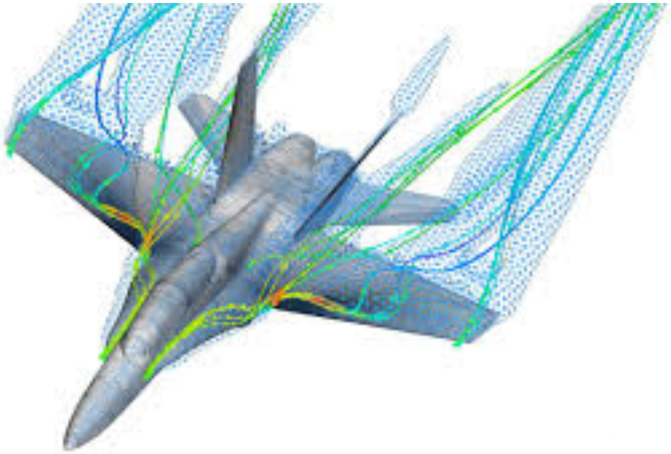
IPSC Node Board

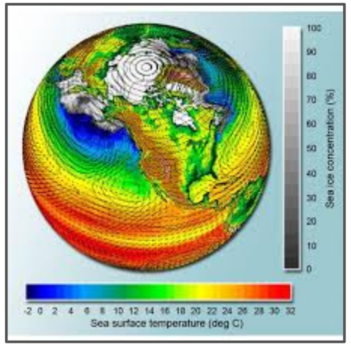
Intel IPSC

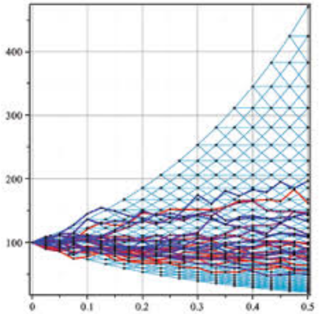
Explosion in New Applications

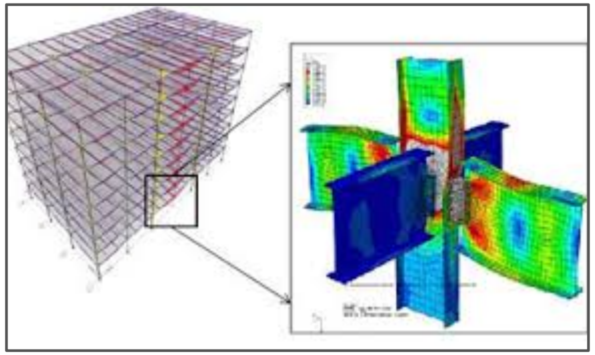
Parallel supercomputers spurred the creation of numerous new applications:

- High energy physics and astrophysics
- Computational fluid dynamics
- Structural mechanics
- Weather simulation
- Climate modeling
- Financial modeling
- Distributed simulation





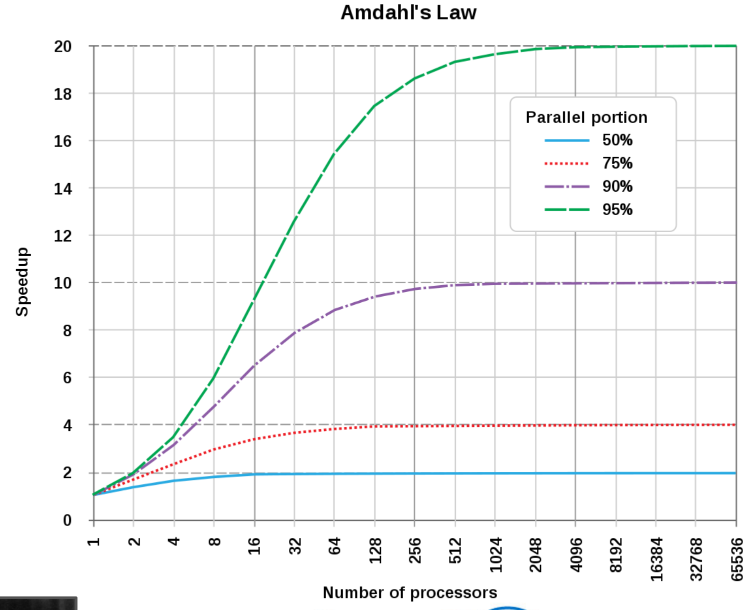


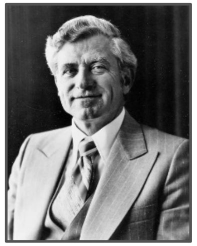


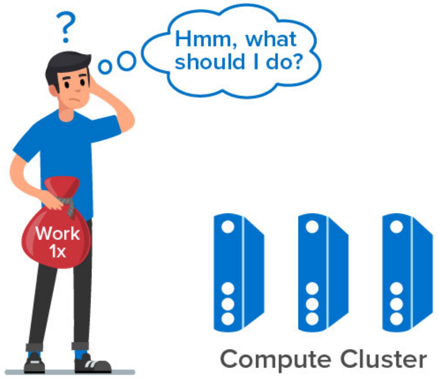
The Challenge: Deliver High Performance

The Goal: Extract parallel speedup on a system with many processing nodes.

- Applications have a combination of parallelizable code and sequential code.
- Sequential code and communication overhead can limit overall performance.
- First described by Gene Amdahl in 1967 (“Amdahl’s Law”) for speedup: S <= 1/(1-p), p = parallel fraction
- Example: If 90% is parallel code, speedup cannot exceed 10X!

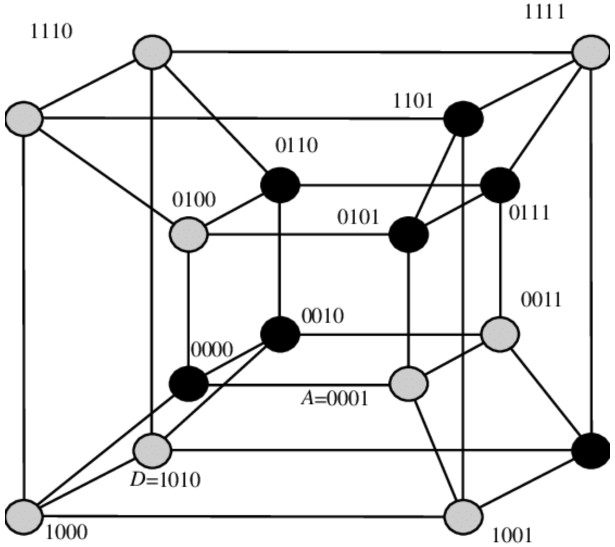




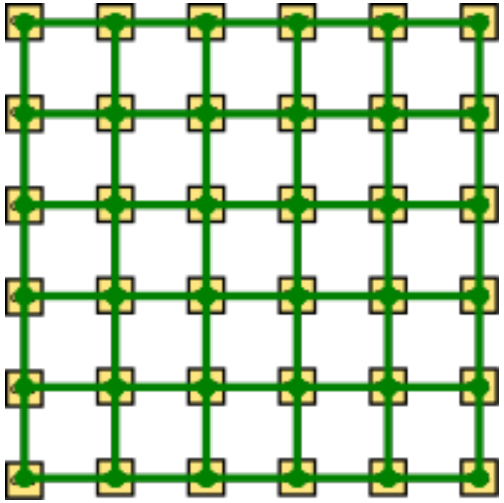


Fast Interconnects Help

They boost throughput and lower communication latency.



4D hypercube



2D mesh with cut-through routing

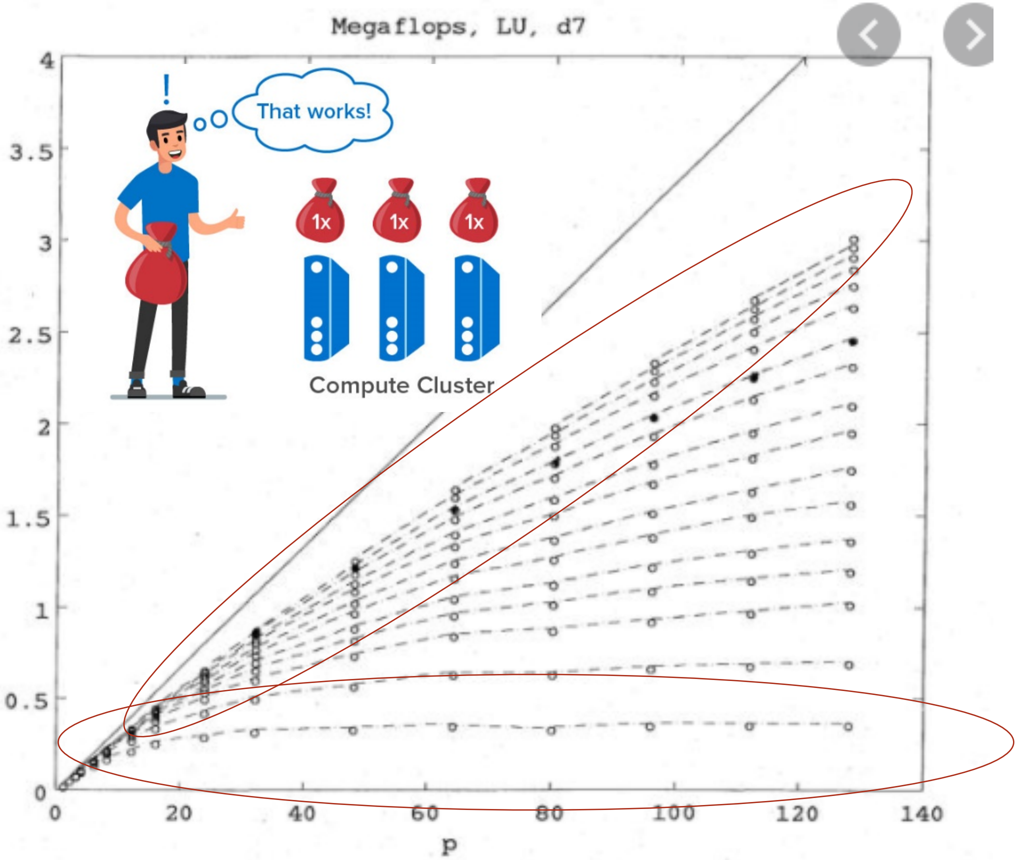
Bill Dally, Caltech

But proprietary networks can be expensive and hard to build.

The Solution: Scale the Workload

Scale the workload to match the system's capacity.

- First observed by Cleve Moler in 1985 while running LINPACK on the iPSC.
- He initially could not get the LU Decomposition algorithm to scale.
- Running the algorithm on a larger matrix hid overheads and extracted higher throughput.
- Moler coined the term "embarrassingly parallel" to describe highly scalable algorithms with low communications overhead.



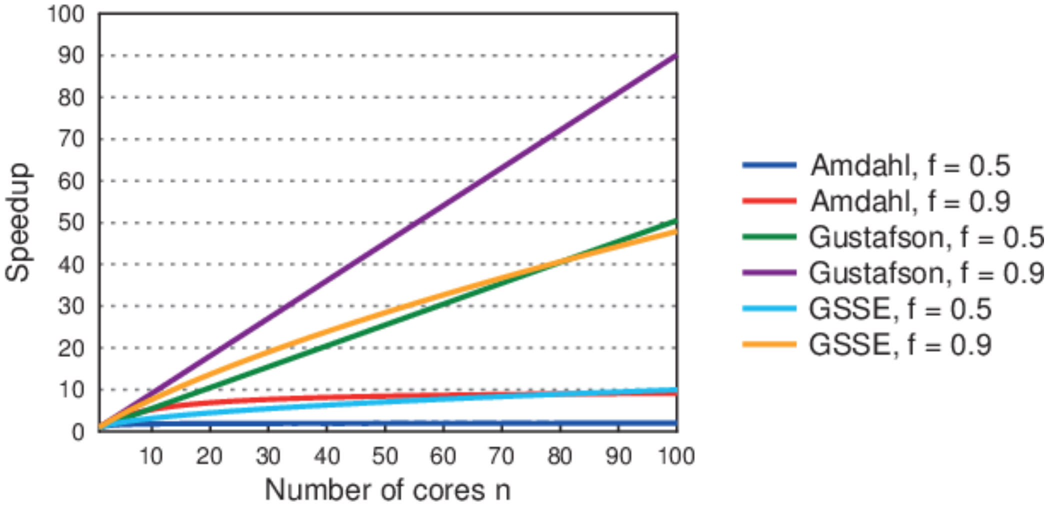
LINPACK throughput for increasing matrix sizes (Moler)

Scalable Speedup is Fundamental to IMC

Scaling the workload maximizes throughput and keeps response times low.

- Quantified by John Gustafson in 1988 (“Gustafson’s Law”): $S = 1 - p + Np$
- Used by in-memory data grids for both distributed caching and parallel computing
- Requires balancing resource usage: CPU, memory, network

Comparison of Amdahl to Gustafson speedup (Juurlink)

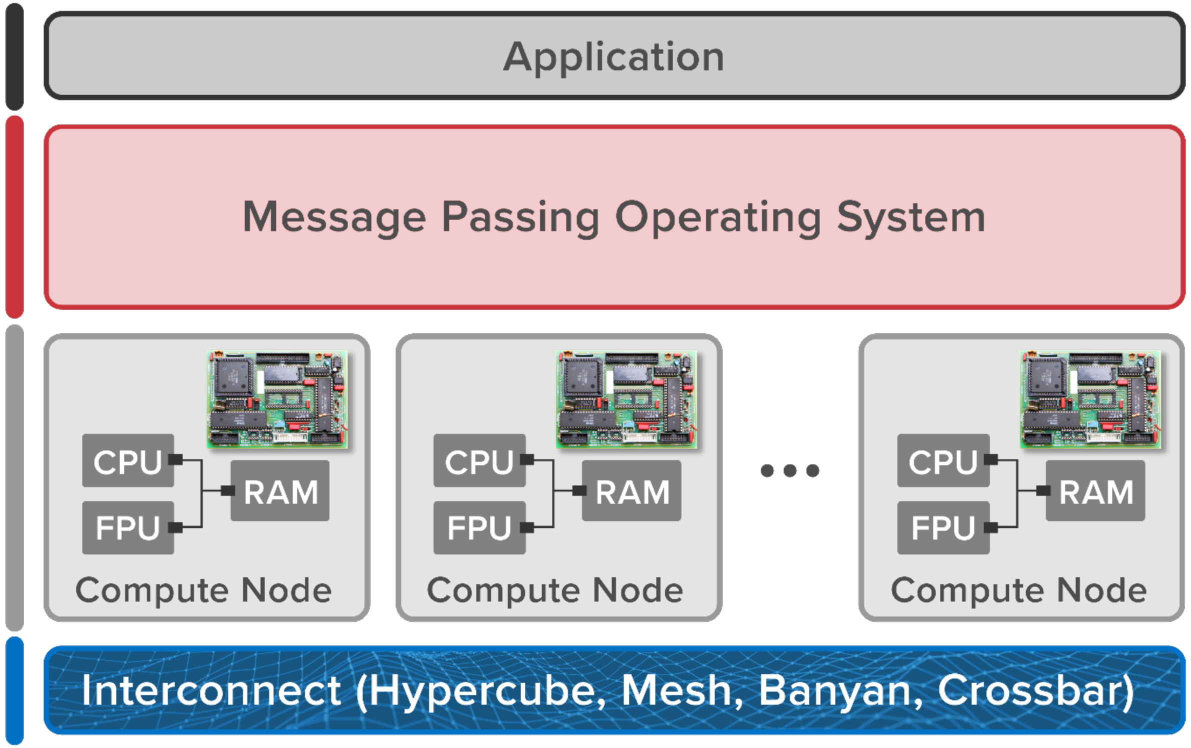


Evolving IMC Architectures

IMC architectures have evolved to take advantage of new technologies.

Original supercomputer architecture

(1980s-1990s)

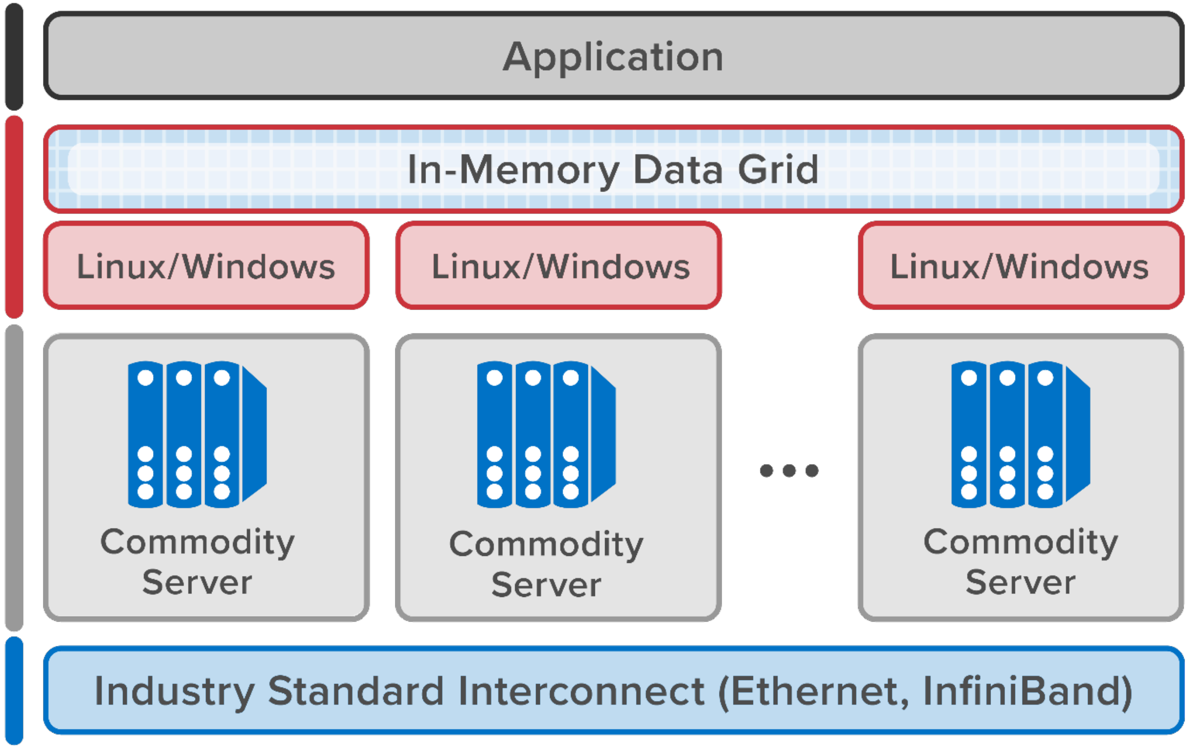


Evolving IMC Architectures

IMC architectures have evolved to take advantage of new technologies.

Compute cluster with in-memory data grid on physical servers

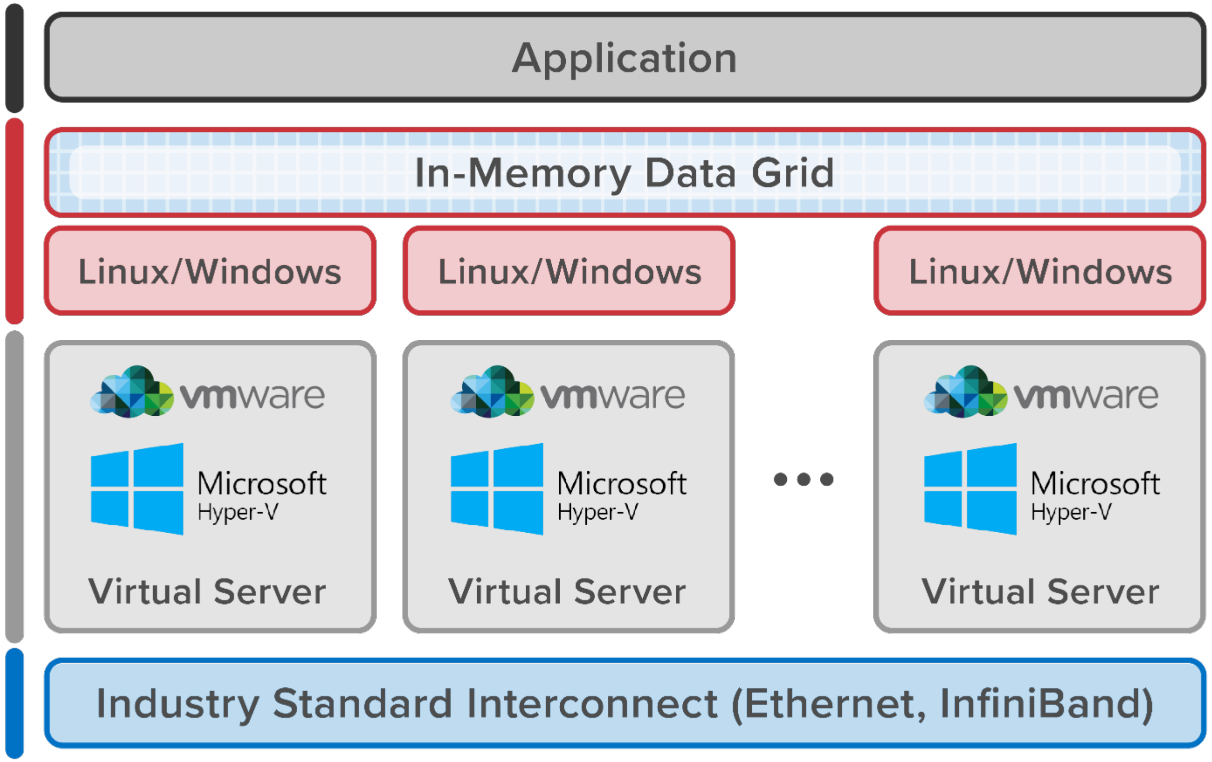
(1998-2001)



Evolving IMC Architectures

IMC architectures have evolved to take advantage of new technologies.

Compute cluster with in-memory data grid on virtual servers (2005)

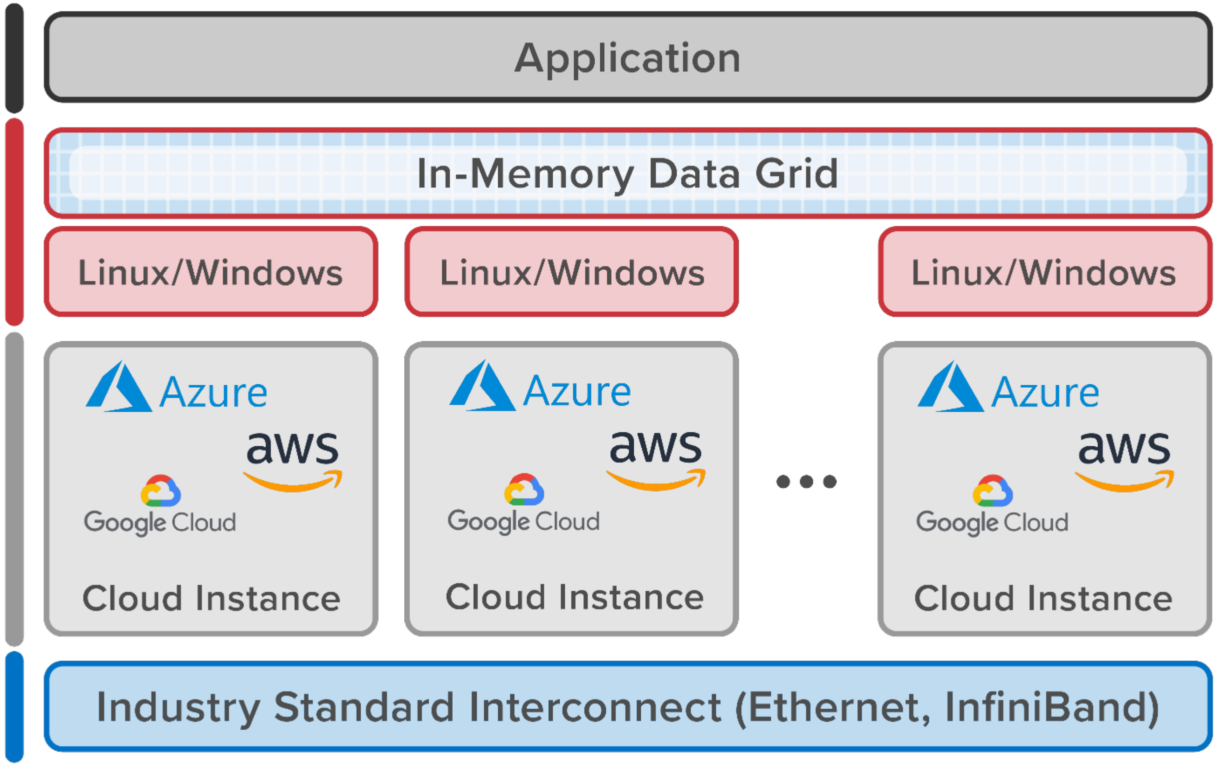


Evolving IMC Architectures

IMC architectures have evolved to take advantage of new technologies.

Compute cluster with in-memory data grid in the cloud

(2007)

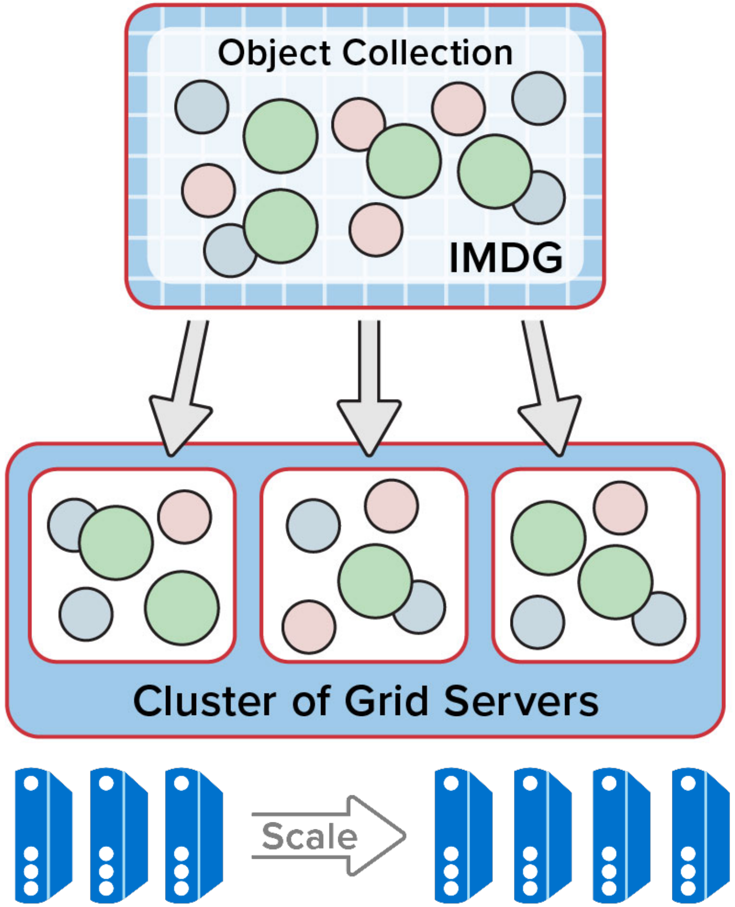


Simplifying the Developer's Task

In-Memory Data Grid (IMDG) provides important software abstractions.

- Message passing is fast but challenging.
- IMDGs were originally developed in 2001 for distributed caching as middleware software.
- IMDG hides complexity:
  - Gives applications a global view of stored objects.
  - Incorporates transparent scaling & high availability.
- IMDG can deliver scalable speedup to ensure fixed response times with growing workloads.

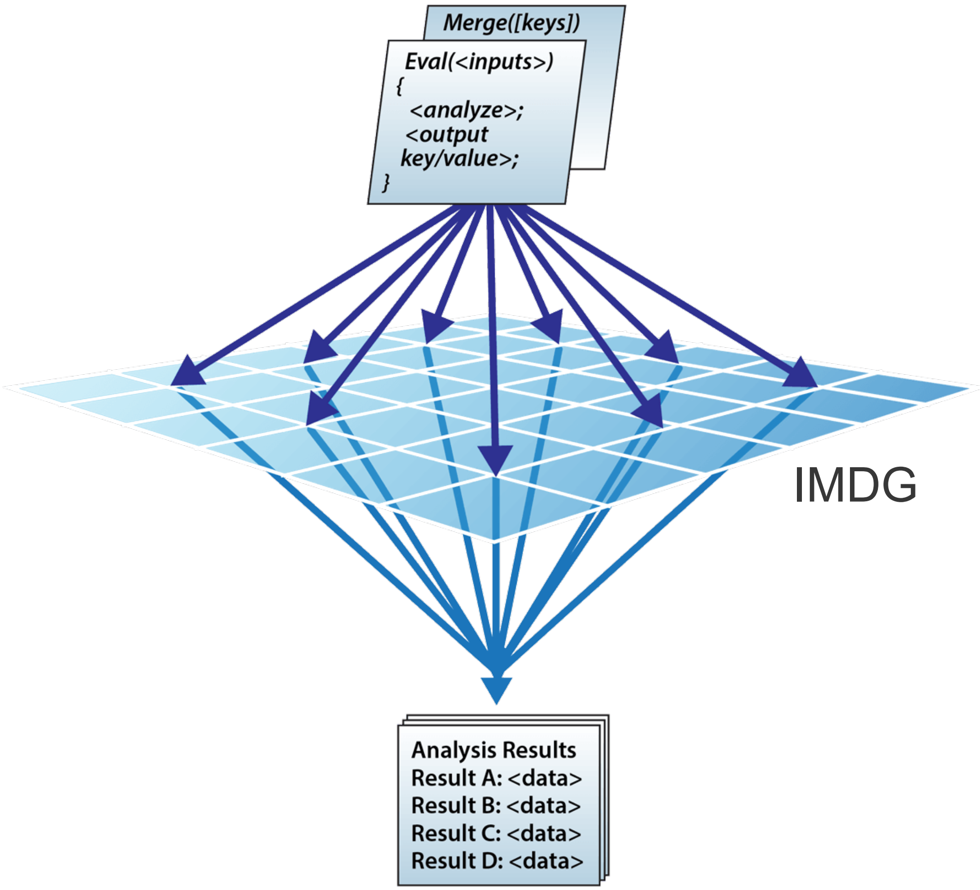
Cameron Purdy, Tangosol



IMDGs Can Host Data-Parallel Computing

Object-oriented APIs enable data-parallel analytics on live data.

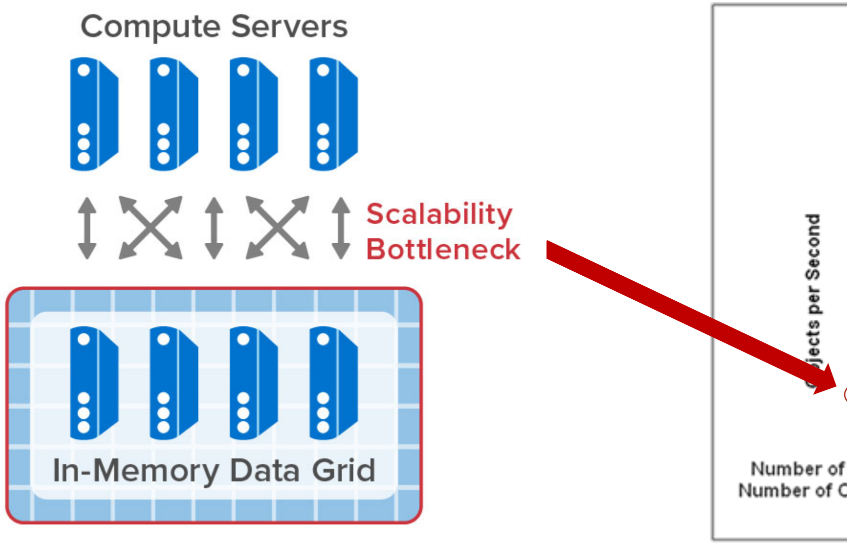
- Example: parallel method invocation (PMI):
  - Run a user-defined method on all objects in a namespace.
  - Merge results and returns them to application.
- Matches semantics of message passing OS in parallel supercomputers.
- Operates on objects stored in the IMDG.
- Provides scalable speedup; reduces network use.

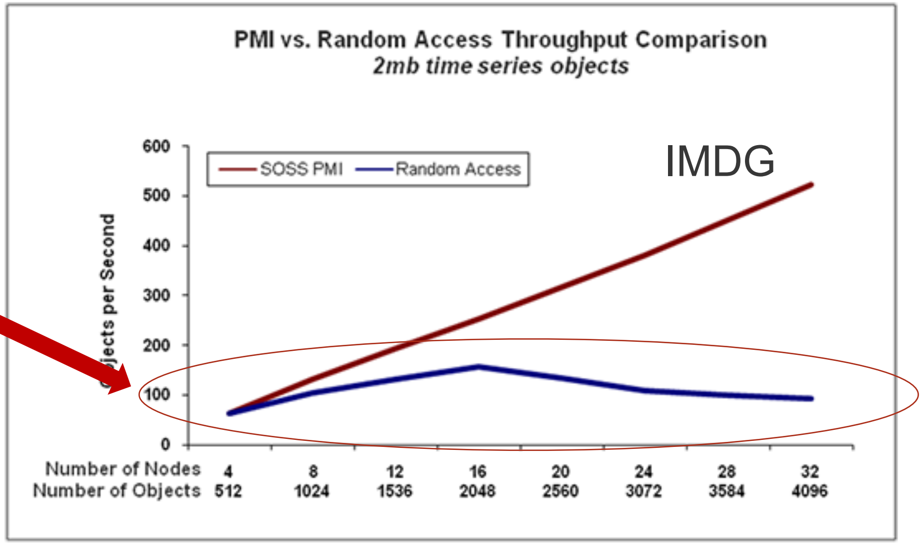


Compute Where the Data Lives

Run parallel algorithms in the IMDG – not on external servers.

- External compute clusters have higher network overhead which lowers scalable speedup.
- IMDGs have scalable CPU resources and can access data without network overhead.





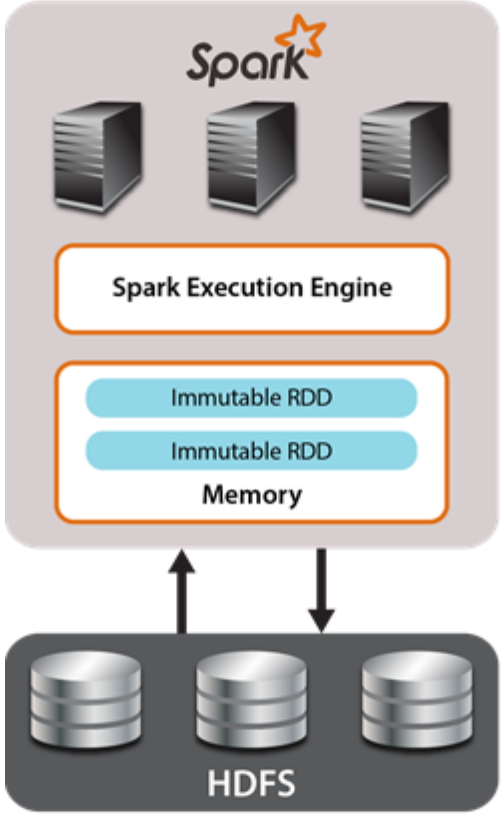
Financial services computation (stock back-testing)

Specialized IMC Software for Big Data

Spark (2009) provides a powerful software platform for parallel processing of large data sets.

- Offers data-parallel operators (e.g., Map, Reduce, Filter) in Scala and Java.
- Uses specialized data sets (RDDs) to host in-memory data.
- Designed for analyzing "big data"

Matei Zaharia, UCB



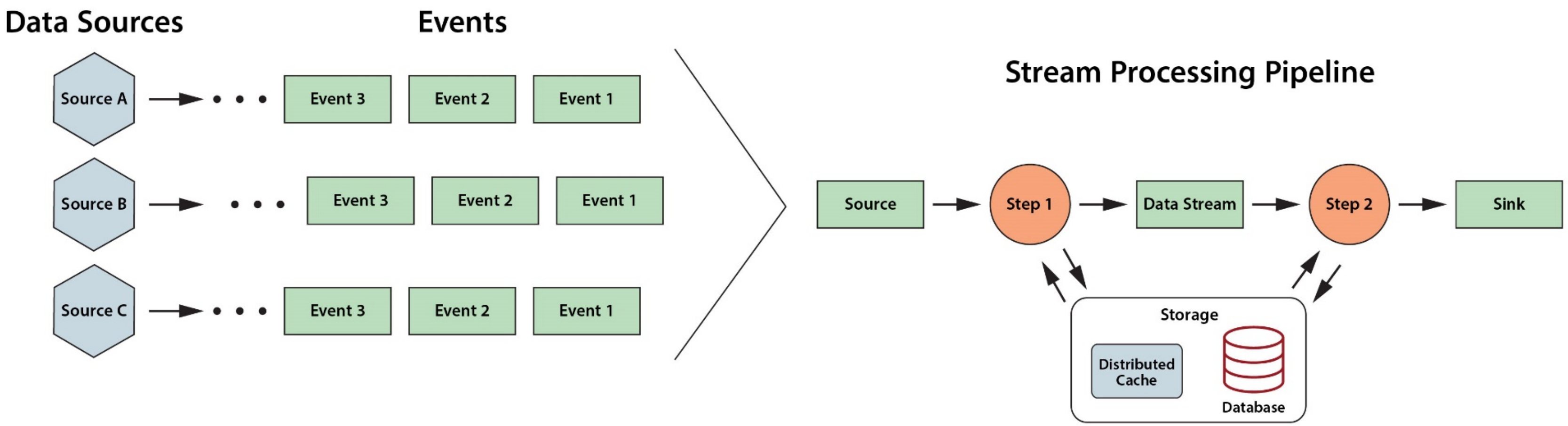
Compare to using an IMDG for "live" data:

- IMDG manages fast-changing data in a key/value store.
- Designed to provide "operational intelligence" in live systems

Scaling Streaming Analytics

The Challenge: Ensure scalable speedup for streaming applications.

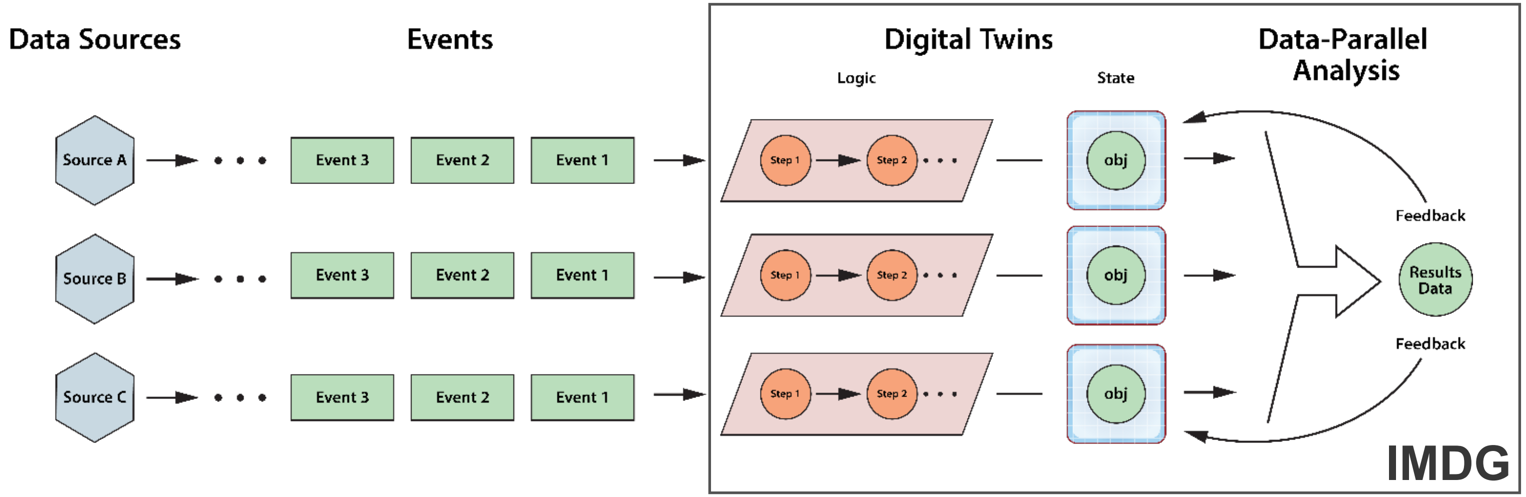
- Typical streaming architectures use a pipelined approach (Storm, Flink, Beam).
- Pipelined streaming applications can be hard to design for transparent scaling.
- They also can encounter network bottlenecks accessing contextual data.



Using an IMDG for Streaming Analytics

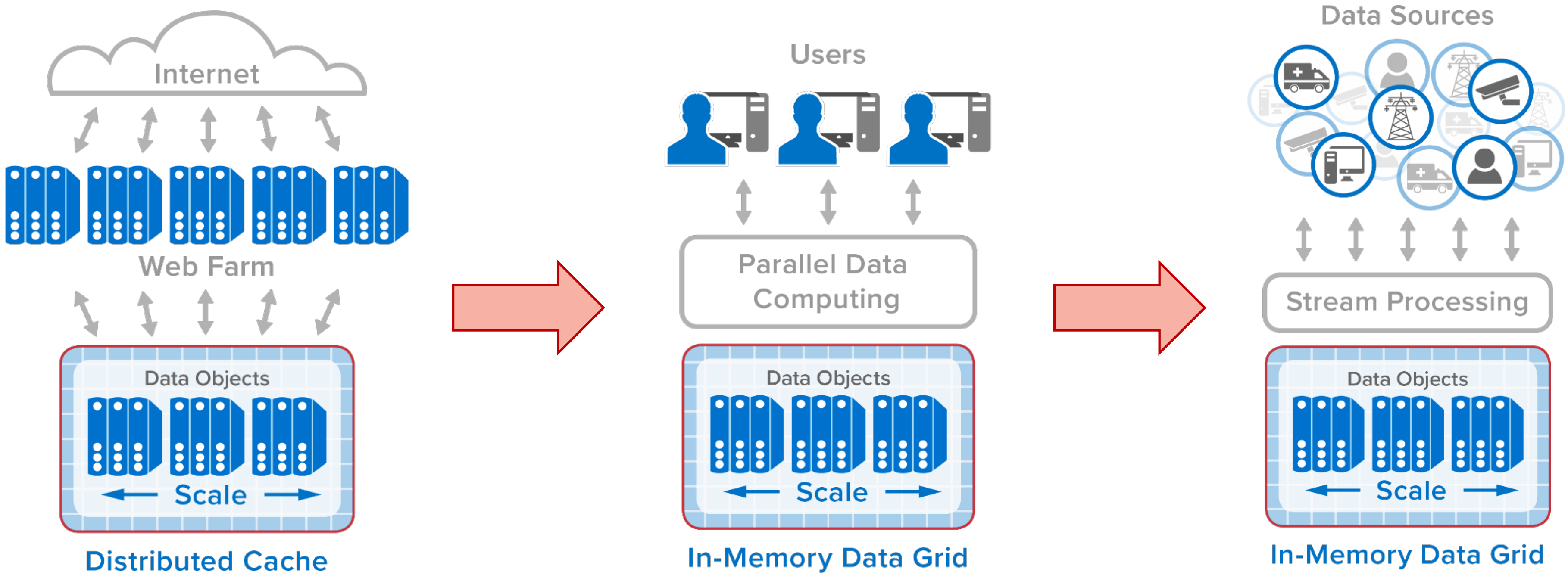
The digital twin model helps ensure scalable speedup.

- Digital twins map cleanly to an IMDG and track data sources using in-memory objects.
- They enable transparent scaling and avoid data motion.
- They also allow integrated, data-parallel analysis.



From Caching to Streaming Analytics

As IMDGs have evolved, scalable speedup drives design choices.



ScaleOut releases: 2005 2008 2017

Takeaways

- In-memory computing has a long, storied history.
- The concept of scalable speedup drives performance and underlies key design choices.
- Maintaining speedup requires balancing CPU, memory, and networking as technology shifts.
- Live systems continue to grow and generate more data to manage and analyze.
- In-memory computing serves as a key technology for extracting value from both live and big data.