YarcData™
A DIVISION OF CRAY INC.


# Introduction to the uRiKA Graphical Database System

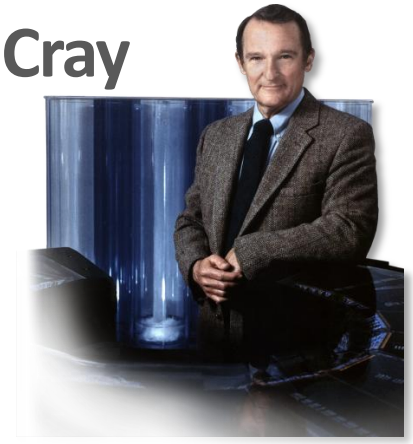
David Mizell
November 11, 2012
SSWS + HPCSW Workshop

# Outline

- What's uRiKA? What's YarcData?
- What are our basic assumptions? (Some are probably different from yours)
- What's different about our hardware platform?
- What's the software architecture?
- So, is it fast?
- Where are we going with this system, technically?
- Who cares?

# Cray? They're still in business??

- Cray Research founded in 1972 by Seymour Cray

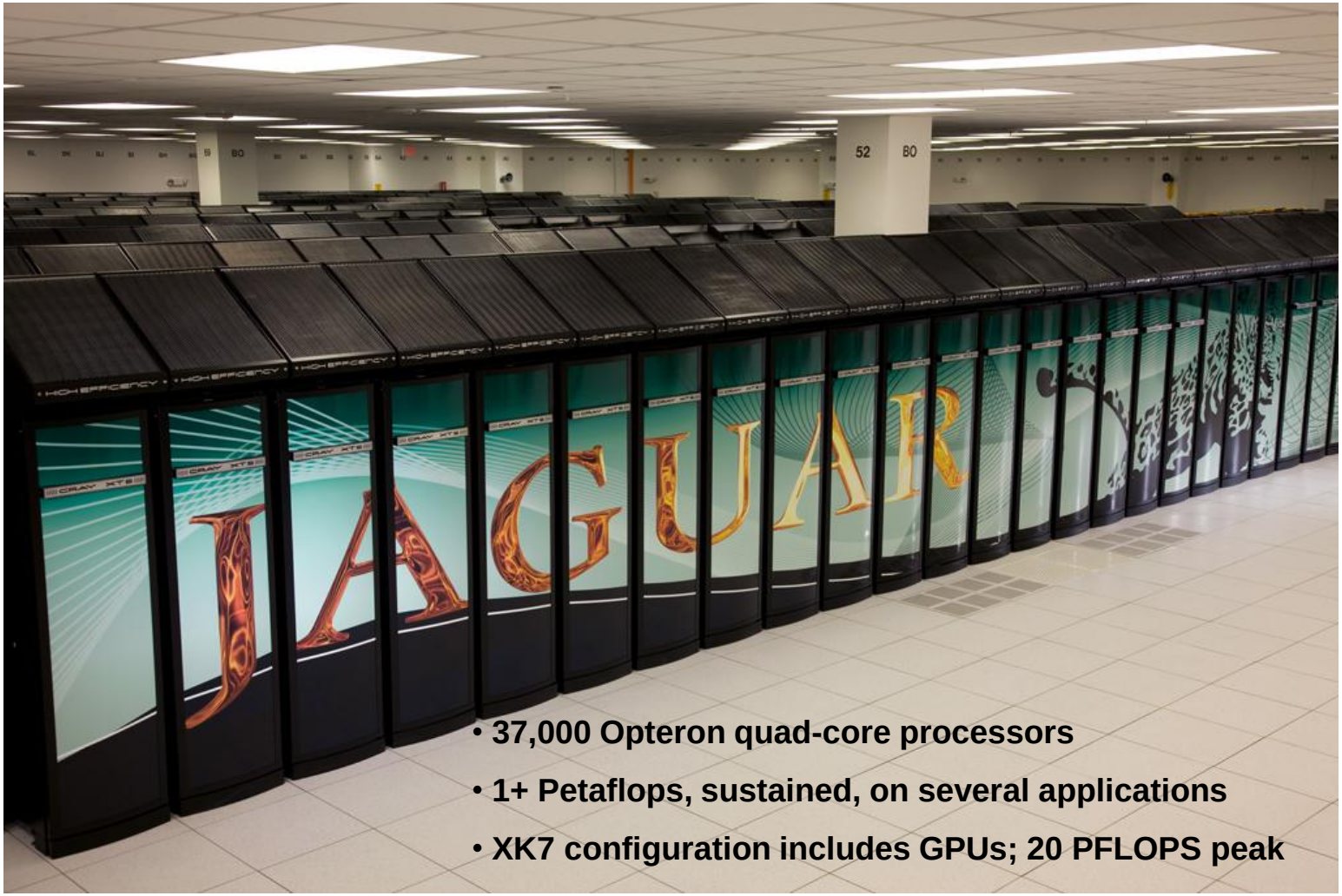


- Bought by SGI in 1996, sold to Tera in 2000;

Tera changed its name to Cray Inc. Until then, Tera was developing an exotic multithreaded supercomputer…

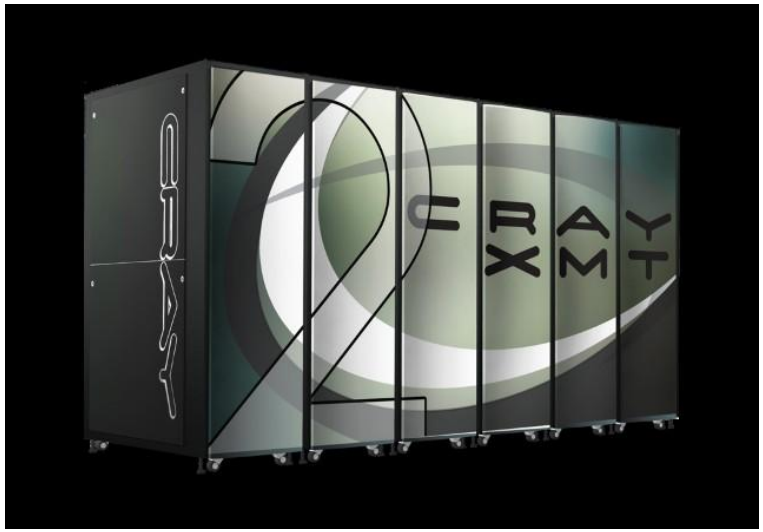
- Cray's main-line product: big distributed-memory supercomputers.

# Like this one: “Jaguar,” Cray XT5 at Oak Ridge National Laboratory, Tennessee. Started out as an XT5; now being upgraded to an XK7, renamed “Titan”



# What's a uRiKA?


"universal RDF information Knowledge Appliance": a SPARQL query engine in an XMT2




Based on the XMT2 "e**X**tremely **M**ulti**T**hreaded system.


Uses the XT5 cabinet/board/interconnect infrastructure.

# …and YarcData?

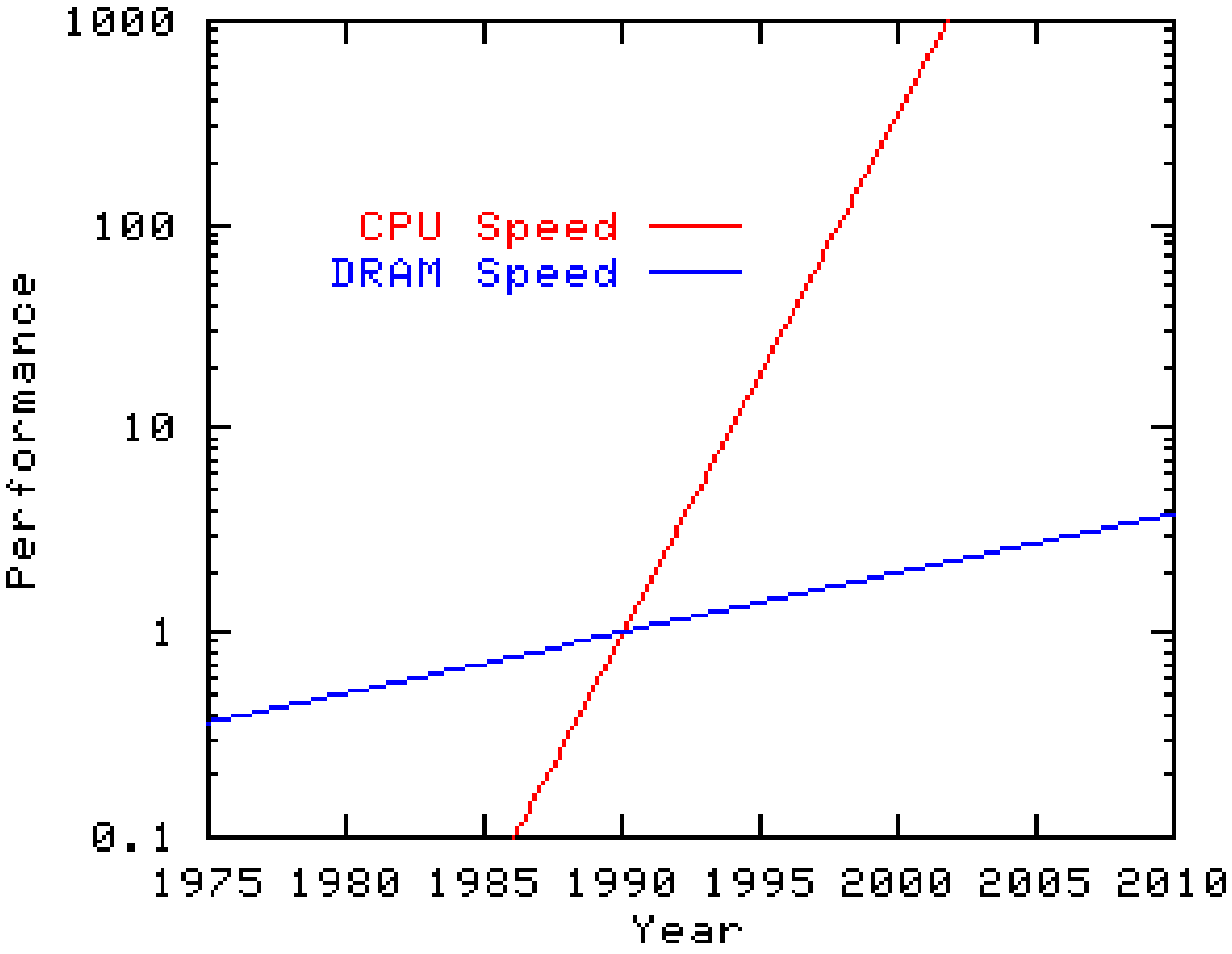

A subsidiary of Cray, focused on marketing the uRiKA system.

# Assumptions…

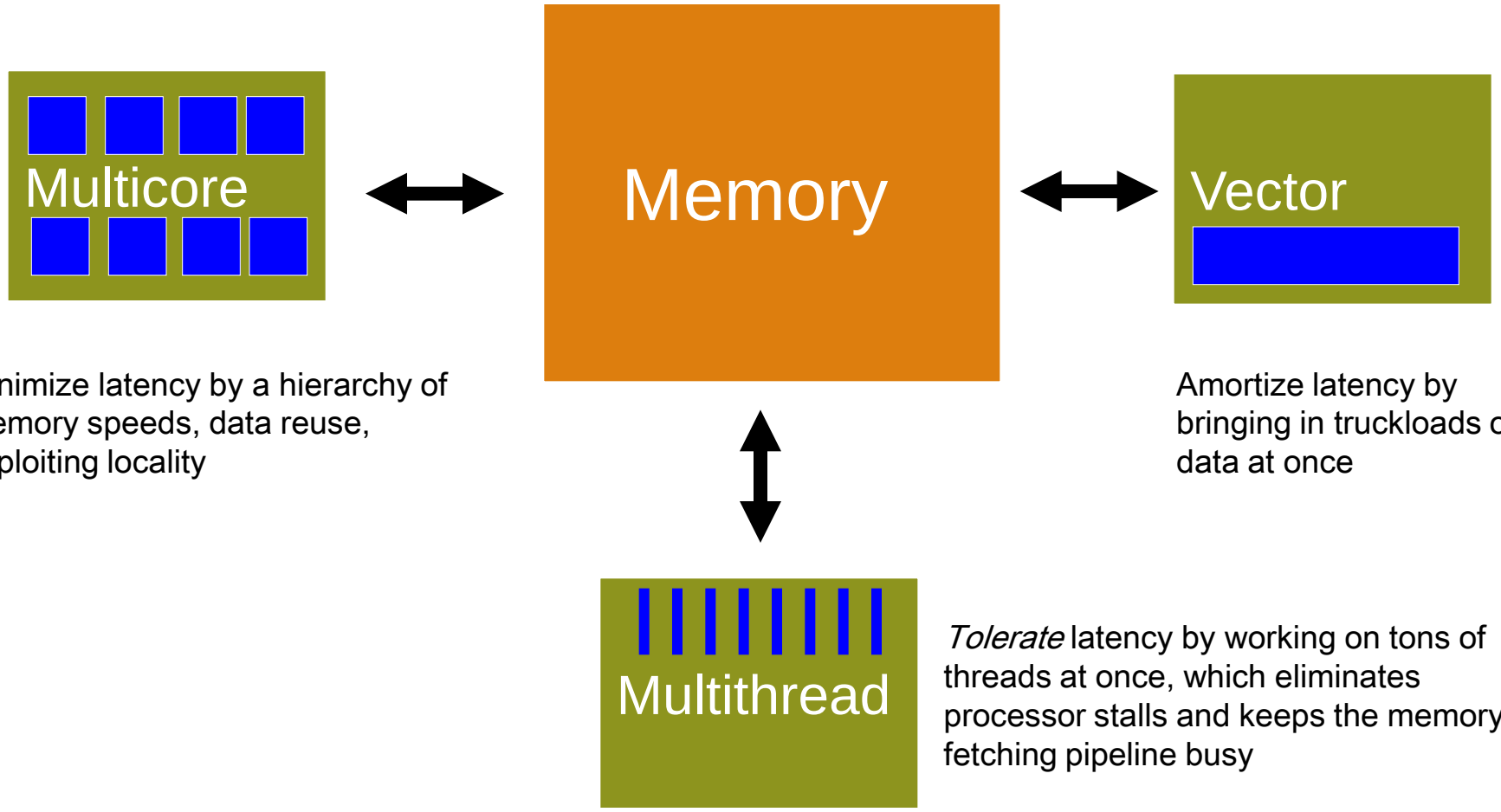
- **…that you and I may share:**
  - SPARQL is a reasonable/useful query language standard.
  - RDF is a reasonable/useful data representation standard.
  - It's interesting that a set of RDF triples defines a directed graph.

- **…and that I make, that you may not share:**
  - It's extremely important – critical – that a set of RDF triples defines a directed graph. Our customers want interactive speed on complex queries against graph-oriented data.
  - The Semantic Web is a third-order consideration at best. The database is in our box. The first-order consideration is to provide fast answers to complex queries against data in our box.
  - We'll begin with the SPARQL standard; we won't end with it.
  - Some of our customers care about ontologies and inference; others don't.

# Moving on to the Hardware Platform: Why a Custom Processor Architecture?



# Ways of Adapting to the Processor/Memory Speed Mismatch



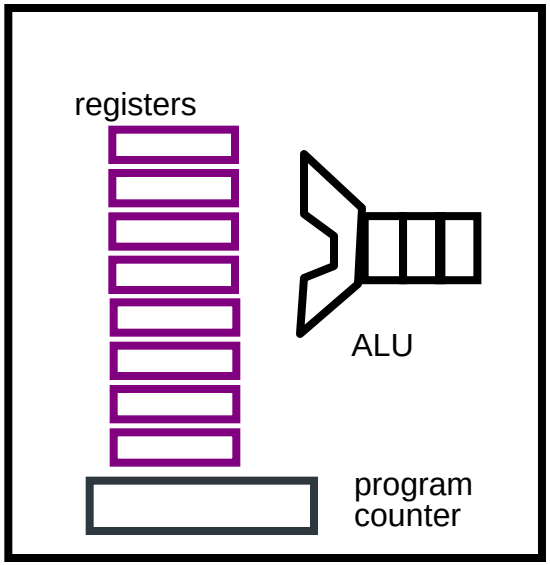

Minimize latency by a hierarchy of memory speeds, data reuse, exploiting locality

Amortize latency by bringing in truckloads of data at once

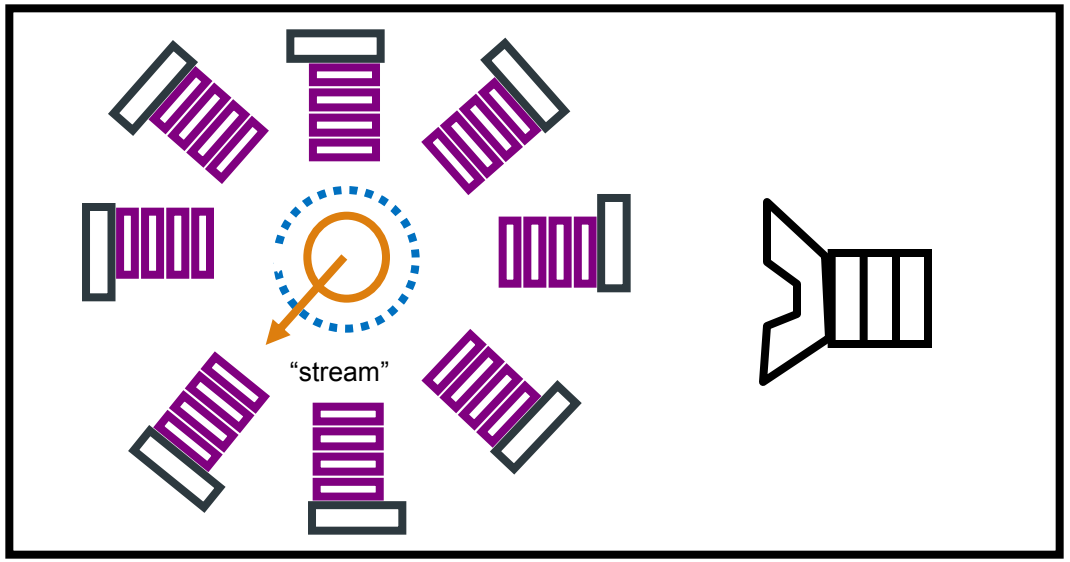
*Tolerate* latency by working on tons of threads at once, which eliminates processor stalls and keeps the memory fetching pipeline busy

# Multithreading

- Many threads per processor core; small thread state
- Thread-level context switch at every instruction cycle



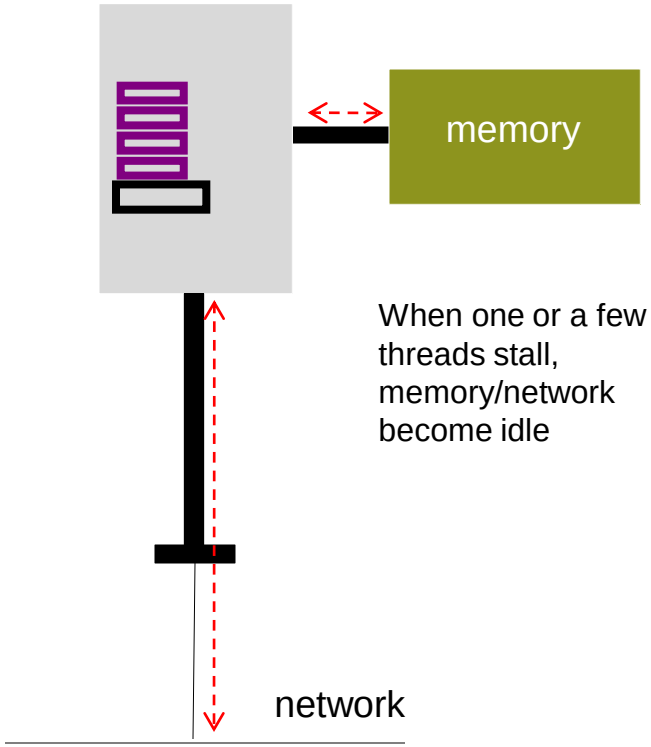

conventional processor



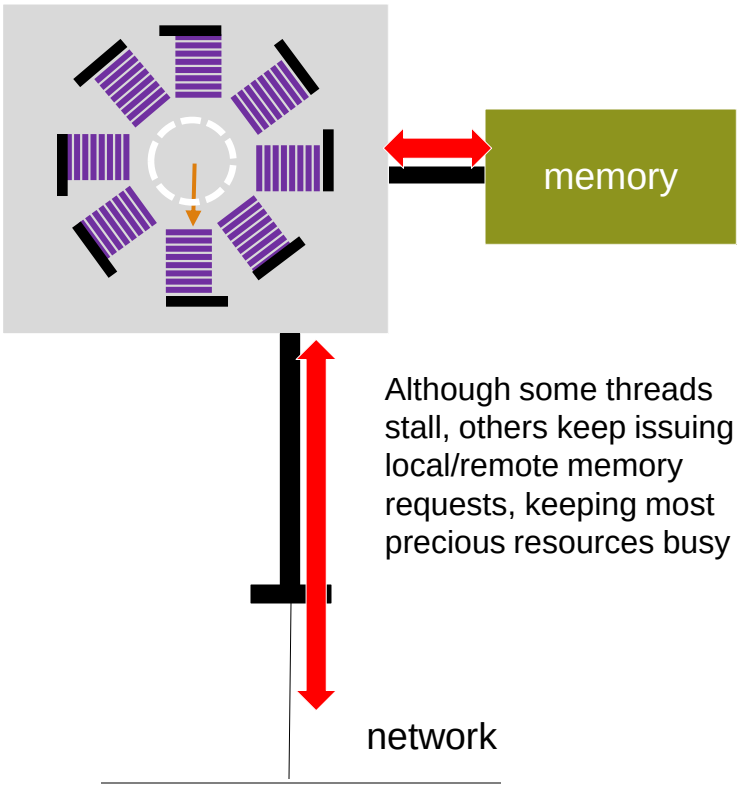

multithreaded processor

# Keeping the Bottlenecks Saturated

- Conventional processor

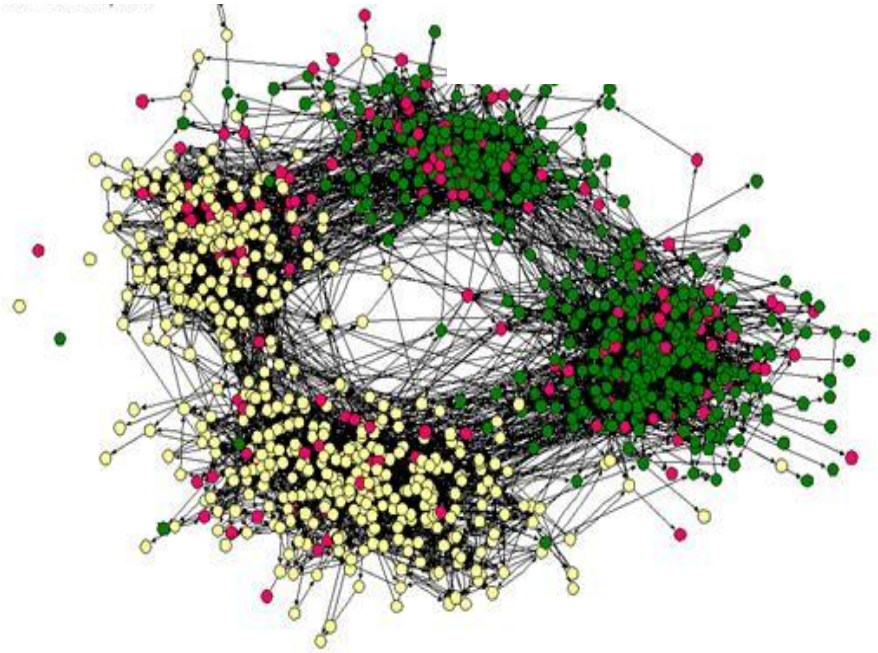



- Multithreaded processor

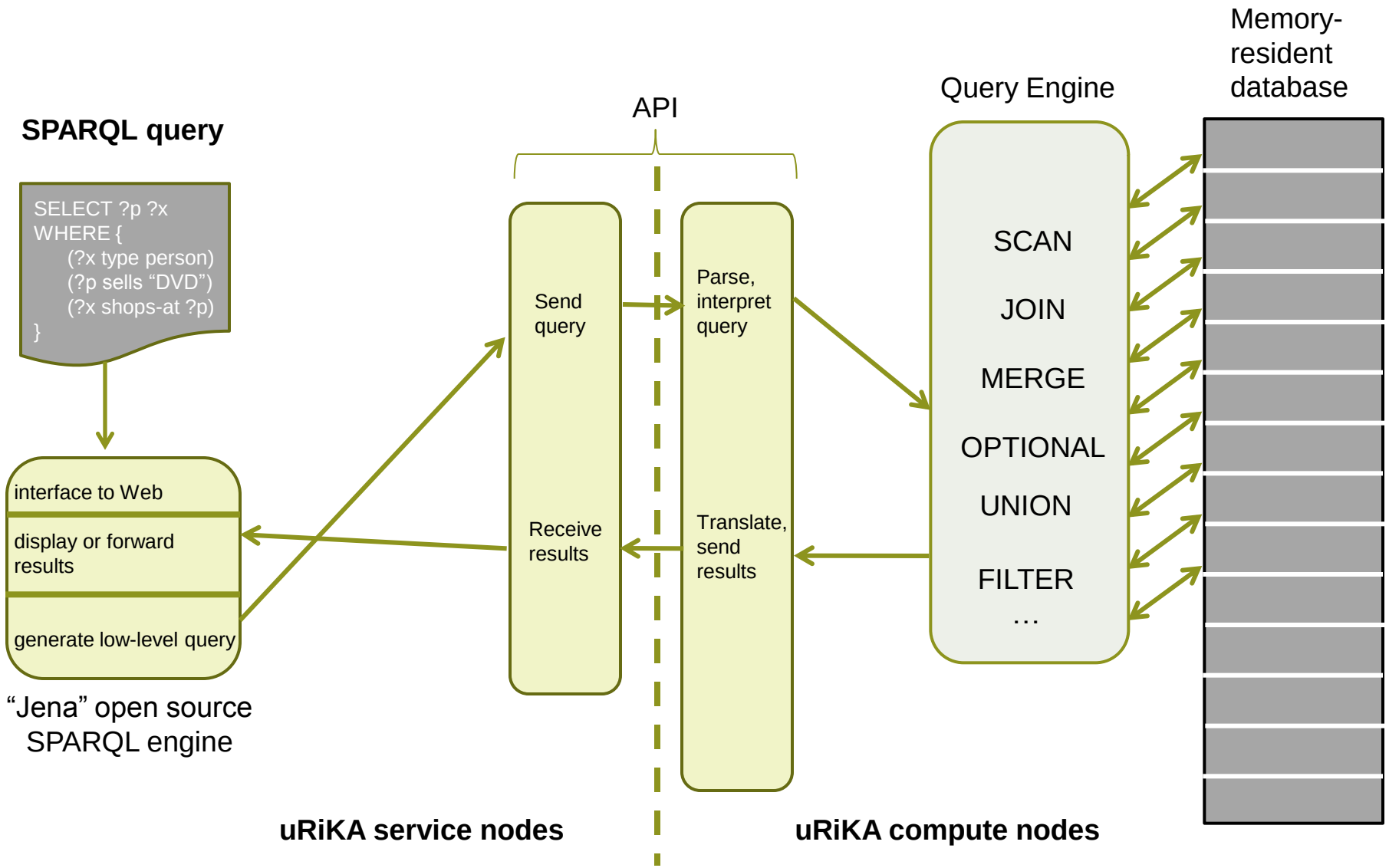


# Multithreading's Ideal Application Characteristics

- **Huge data structures**
  - Too large for one node of conventional system
- **No locality of reference**
  - No way to partition data structure so that most references are local
- **But lots of parallelism**
- **i.e., great big ugly graphs**



# Software Architecture of the Query Engine

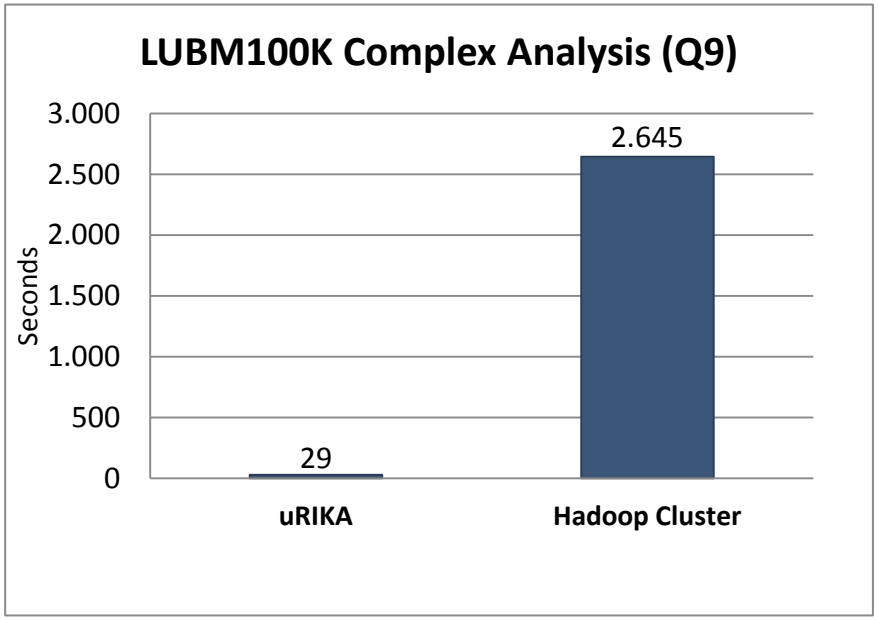


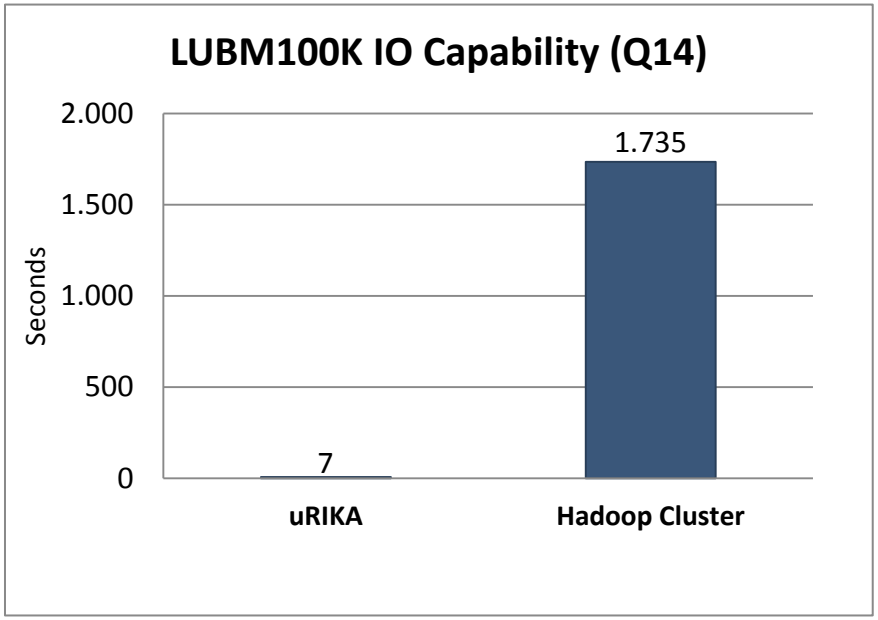
# How the Software Exploits the Hardware

- **uRiKA has a huge, shared memory. We trade space for time almost every time.**
  - Index arrays for every field (subject, predicate, object, subgraph) of the database
  - Use hashing for lookups and comparisons. Hashed dictionary lookups, hash joins
- **Use operations that the hardware and compiler are good at**
  - Scan loops through 1D arrays
  - Write loops that the compiler can turn into recurrences or reductions
- **Use well-established database optimizations**
  - Center on the parts of the query that produced the fewest intermediate results
  - Schedule parts of the query in the order that minimizes work
- **Preprocess as much data as possible at load time**
  - Use lookups rather than computation when processing a query
  - Represent subject, predicate object names as integers
  - Run preprocessing in parallel so that it's still fast

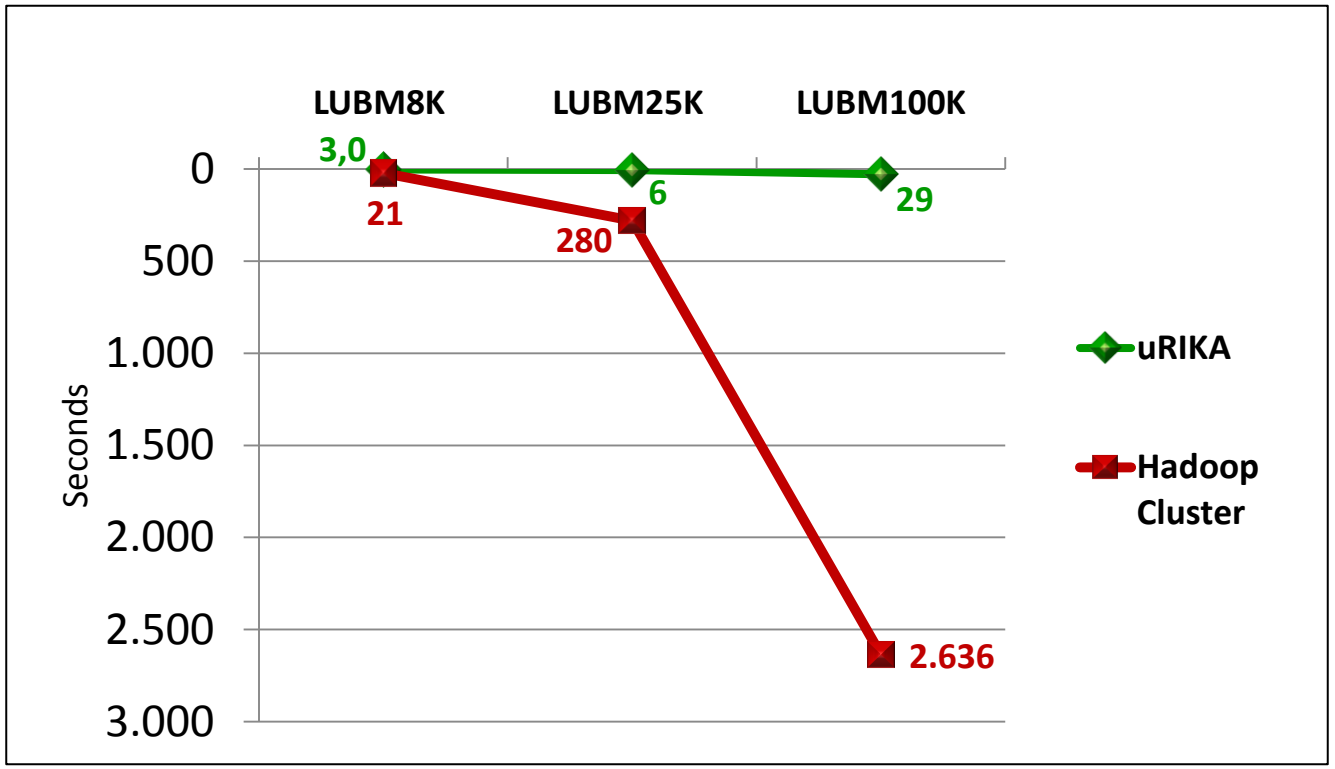
# So, is it fast?

# Performance LUBM100K



# Serious scalability



# Future Directions for the Software

- **SPARQL is pretty good at asking questions about patterns in the data**
  - "Show me any grad student who attends a class taught by his/her adviser, who has co-authored a paper with the adviser, and attended a conference with the adviser and another professor in the same department"
- **It's not so good at asking overall questions about the structure of the graph**
  - "If this graph were looked at as a communication network, through which nodes does most of the communication flow?"
  - "Are there clusters of nodes that connect with each other much more densely than with the rest of the graph?"
  - "Is there a way to get from Node X to Node Y? What's the shortest way?"
- **We are looking at ways to add these "global" graph analysis capabilities to the system.**
  - Existing graph algorithms toolkits, such as the Knowledge Discovery Toolkit
  - Domain-specific graph analysis languages, such as Green-Marl
  - Either would be extended to be able to work with the RDF data items
- **As well as…**
  - Reification
  - Security
  - Dynamic inference
  - etc.

# So who cares?
# Markets we're going after:

- **Intelligence/law enforcement/cyber-security**
  - Some are traditional XMT customers; others want a system based on RDF/SPARQL
- **Health & life sciences**
  - Bioinformatics community has adopted RDF/SPARQL more than any other scientific community
- **Finance/banking**
  - Problems like money laundering, insider trading amenable to graph-oriented queries
- **Traditional HPC**
  - EG new approaches to climate modeling: "teleconnections" – distant places whose weather has large positive or negative correlation

# Thanks! Any Questions?
# Anybody want a job?

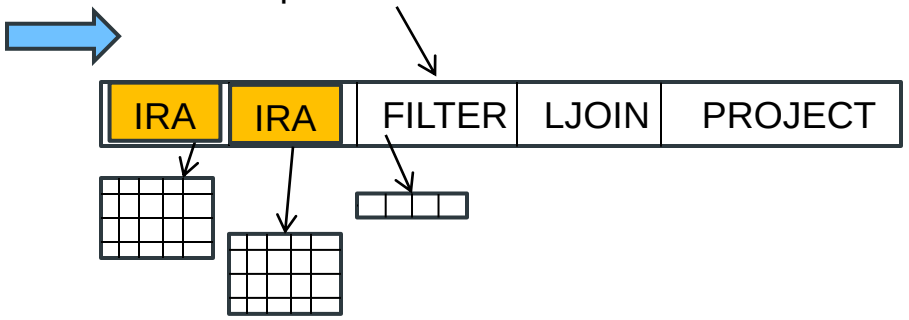
# backup

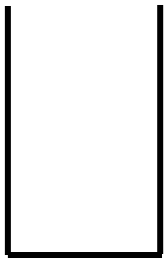
# Database Operations

"SPARQL Algebra" version of query

```
( project  ?x  ?y
  (leftjoin
    (filter ( >  ?y  "1.0"^^xsd:double )
      ( quadpattern
        ( quad  ?x  <pred1>  ?a  )
        ( quad  ?a  <pred2>  ?y  )
      )
    )
    ( quadpattern
      ( quad  ?a  <pred3>  ?b )
    )
  )
)
```

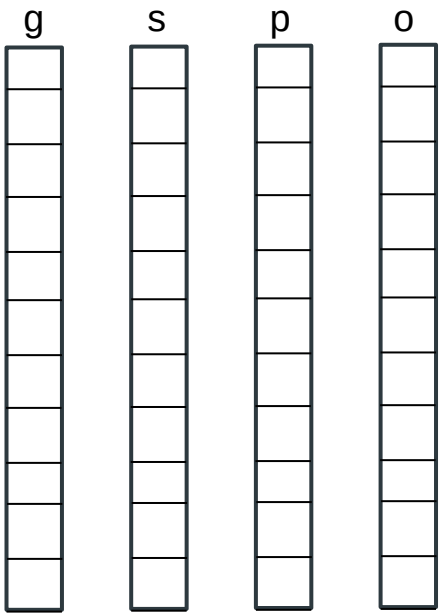
Query gets translated into "Dispatcher List"



intermediate results stack

Index arrays




Database