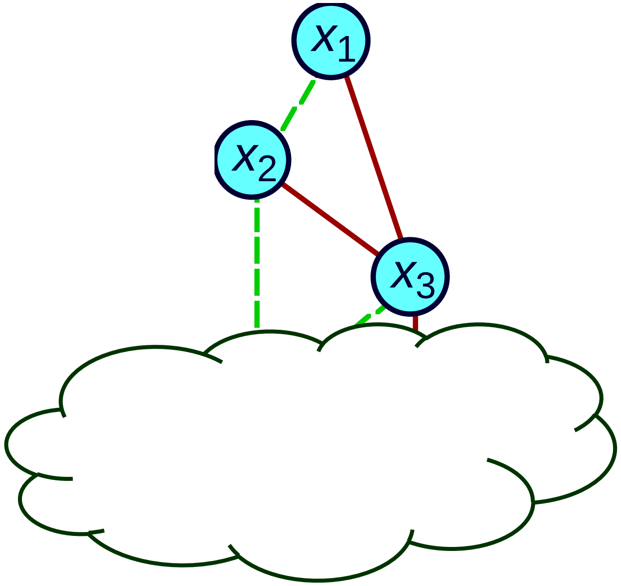


Cloud BDDs

Randal E. Bryant
Carnegie Mellon University

http://www.cs.cmu.edu/~bryant

Ed’s PhD Students

* Will Klieber

* Sicun Gao

Himanshu Jain

Nishant Sinha

* Pankaj Chauhan

* Muralidhar Talupur

* Anubhav Gupta

Alex Groce

* Sagar Chaki

* Dong Wang

* Sergey Berezin

Wilfredo Marrero

* Yuan Lu

* Marius Minea

Vicky Hartonas-Garmhausen

Somesh Jha

Sergios Campos

* Xudong Zhao

* David E. Long

* Jerry. R. Burch

* Ken L. McMillan

M.C. Browne

* David L. Dill

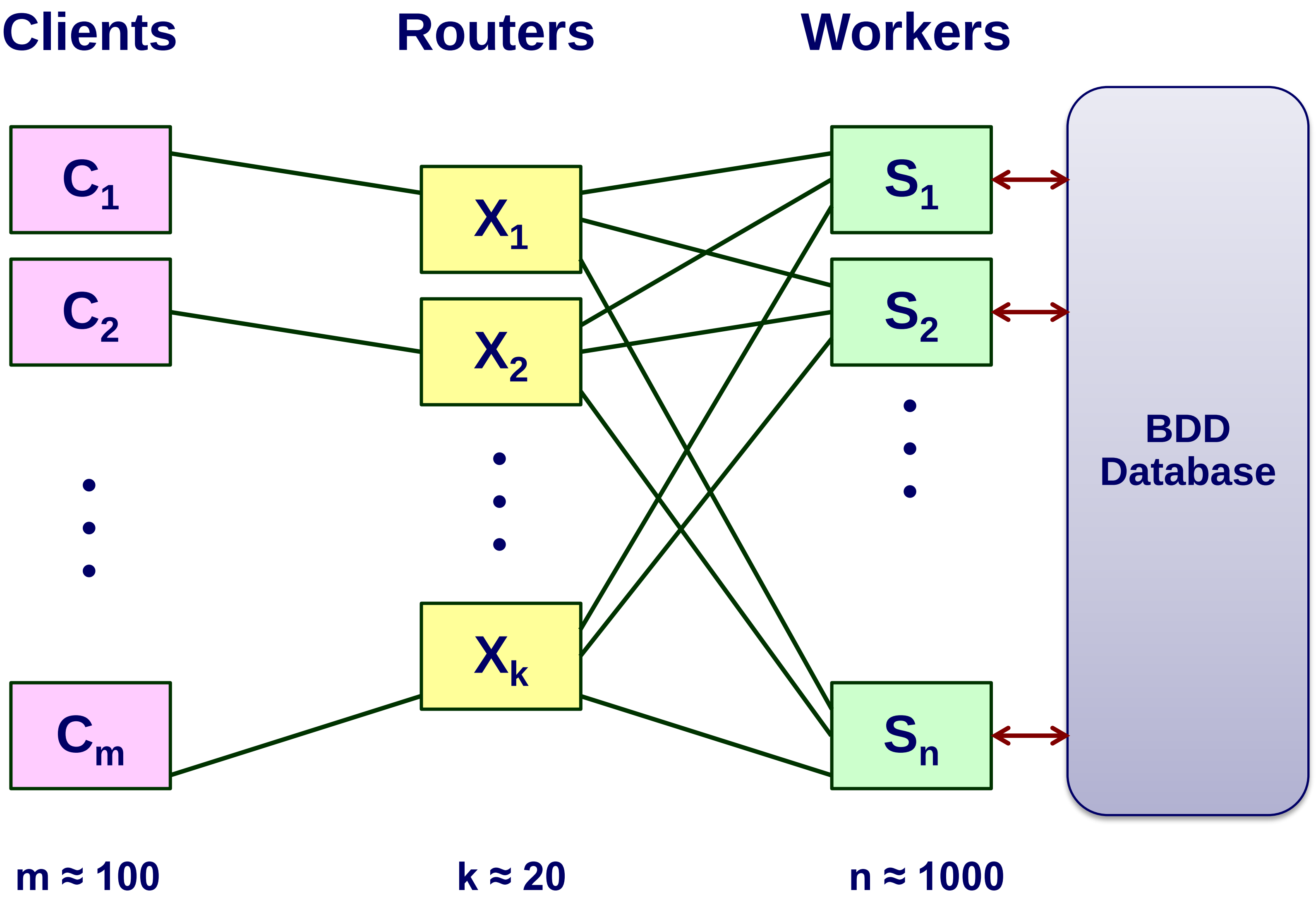
Bud Mishra

A. Prasad Sistla

Christos N. Nikolaou

E. Allen Emerson

Proposed System Structure



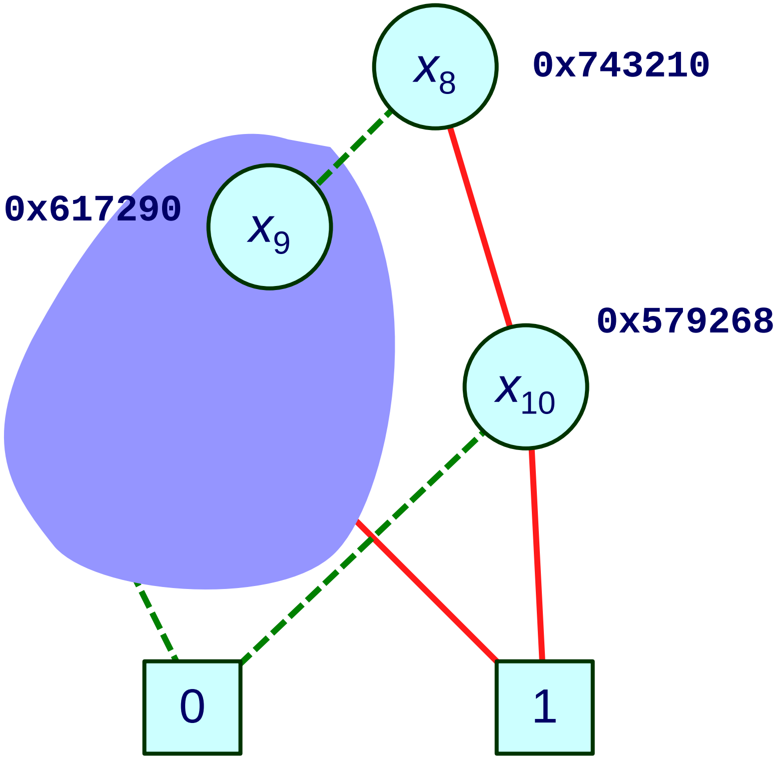
Traditional BDD Representation

0x743210:

8	index
0x579268	Hptr
0x617290	Lptr

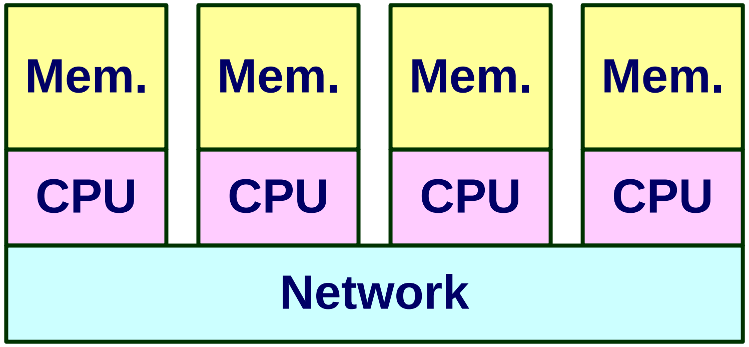
Based on Pointers

- Node represented by address
- Location of information about node



All data within single address space

Shared-Nothing Implementation



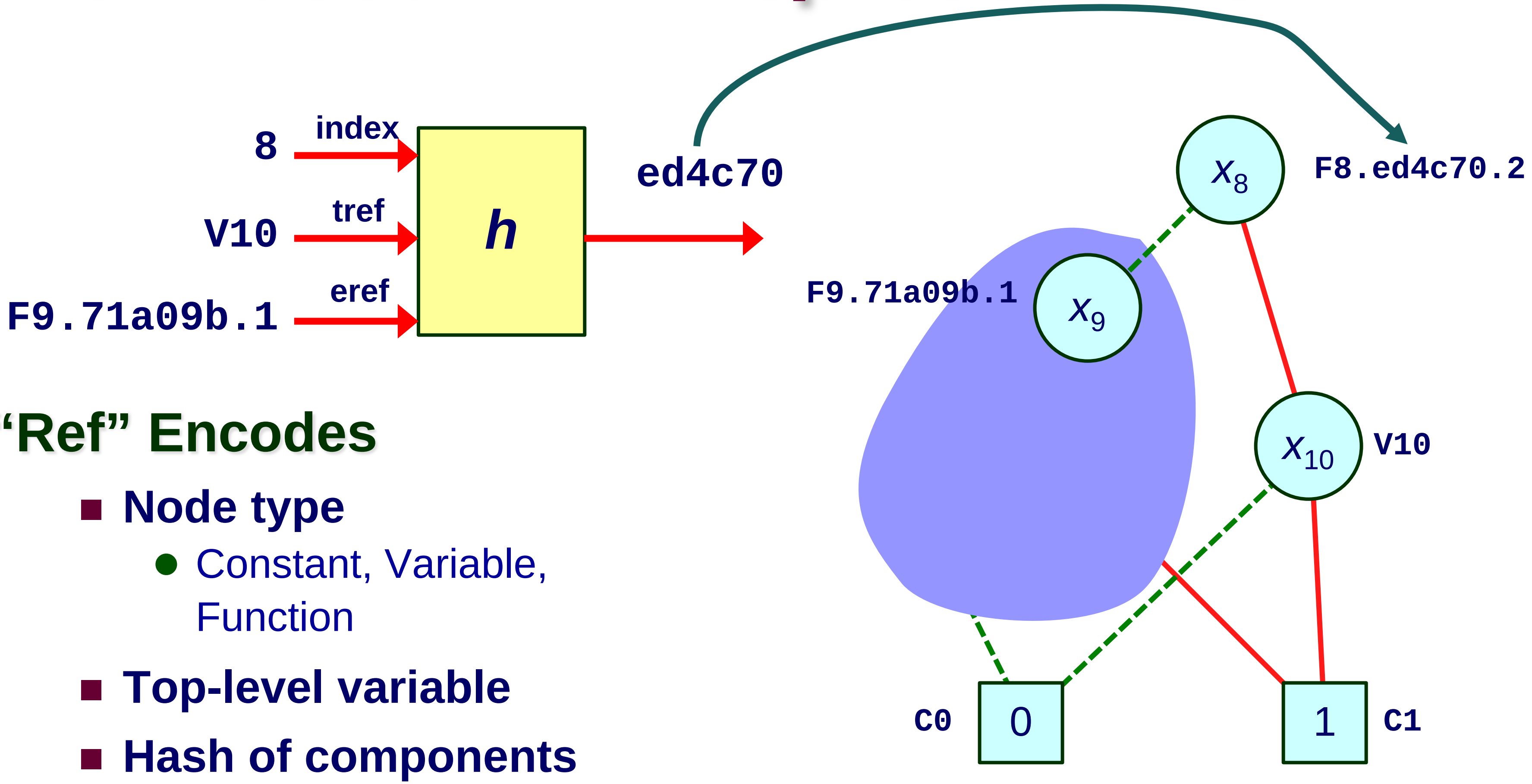
Only Way to Achieve True Scalability

- Large number of low-cost nodes
- Single resource shared by many users

Distribute Data Structures Across Processors

Must find alternative to pointer-based representation

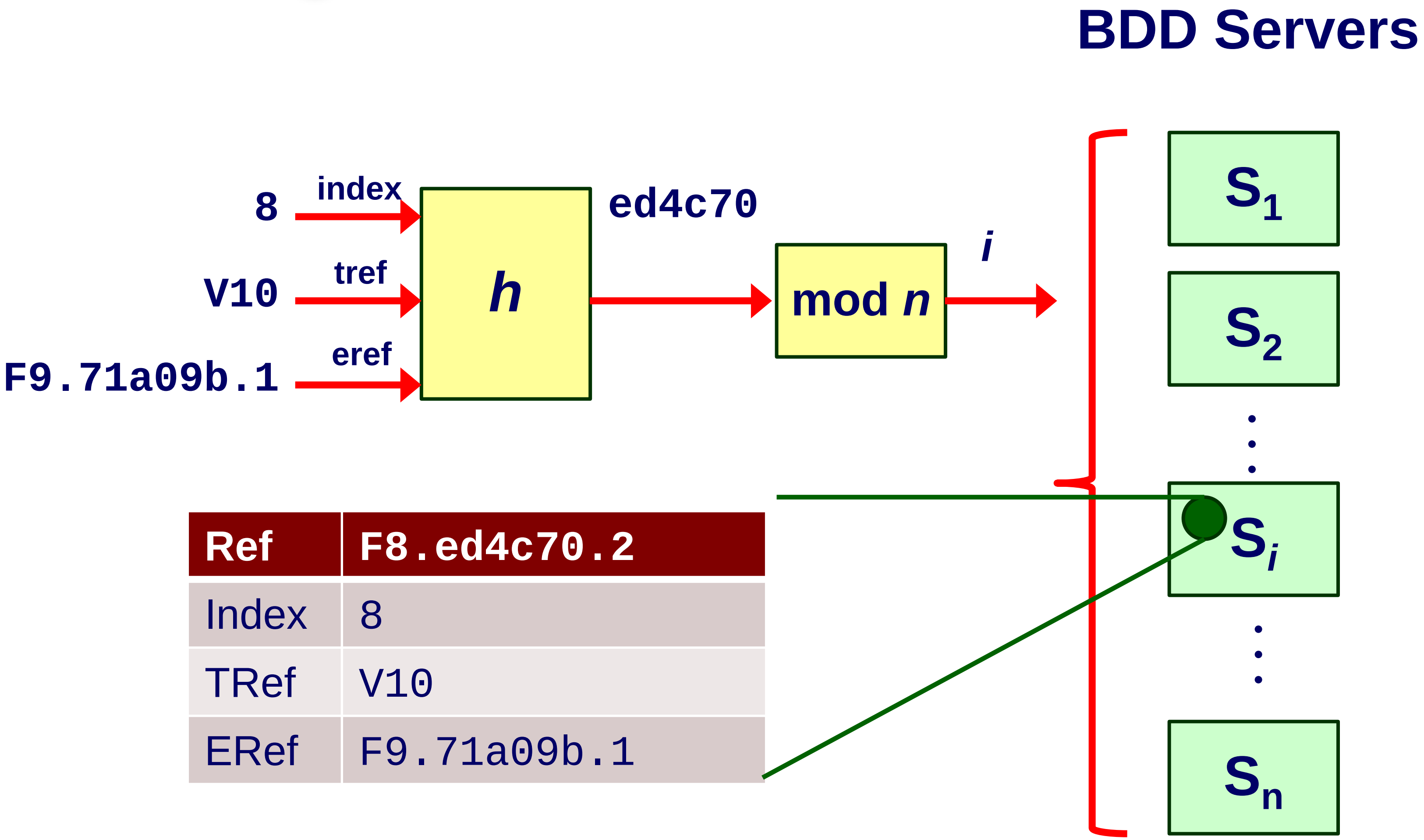
Ref-Based BDD Representation



"Ref" Encodes

- Node type
  - Constant, Variable, Function
- Top-level variable
- Hash of components
- Uniquifier
  - To resolve hash collisions

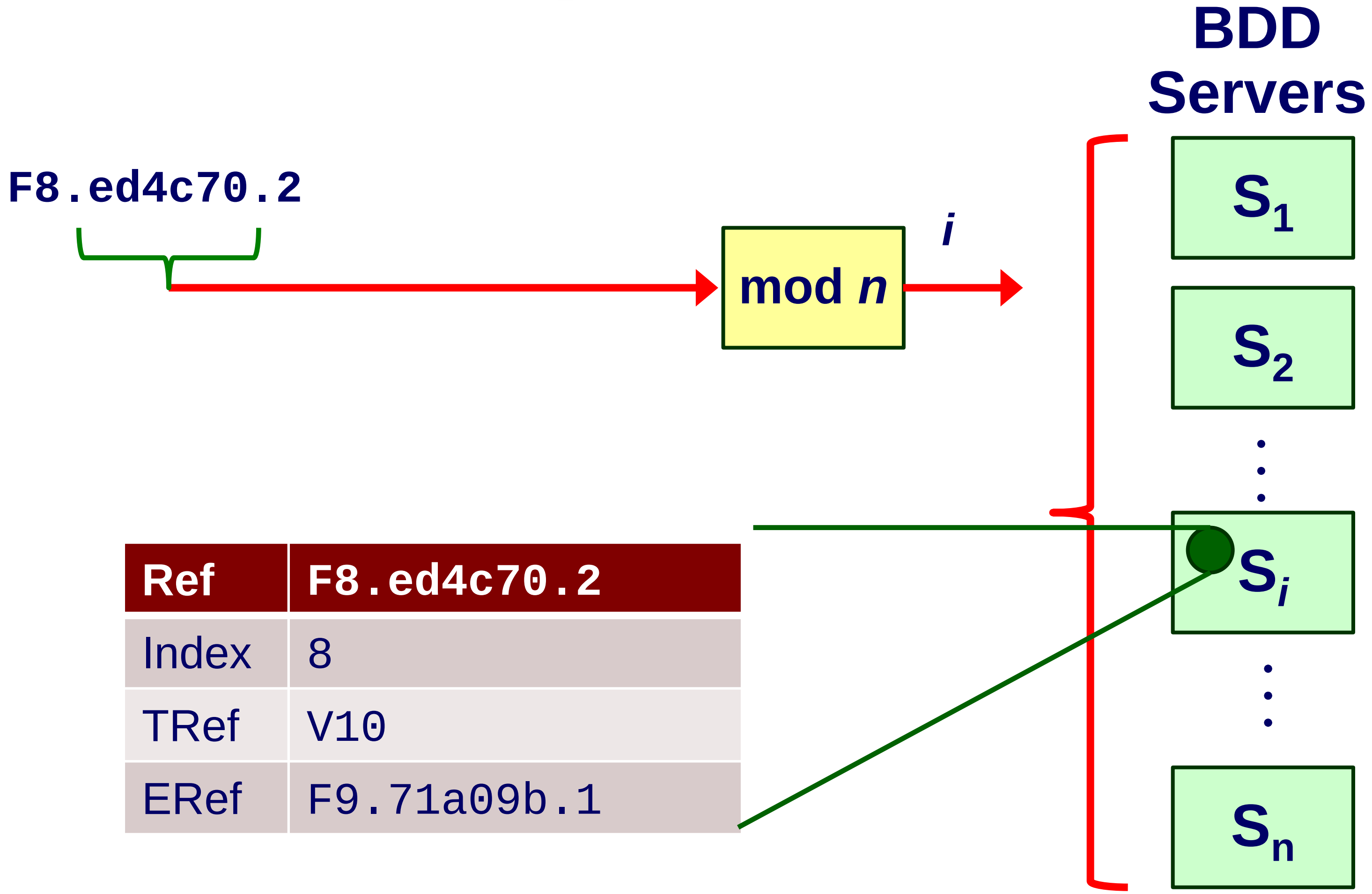
Storing a Ref



Ref	F8.ed4c70.2
Index	8
TRef	V10
ERef	F9.71a09b.1

- **Entry describing node stored according to its hash signature**
- **Unique table distributed across workers according to hash**

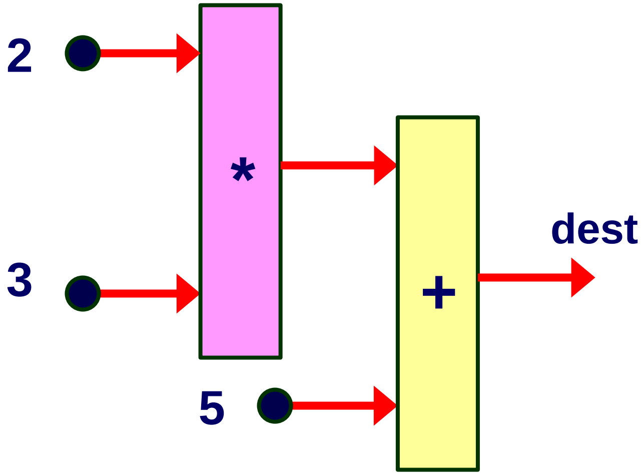
Dereferencing a Ref



Ref	F8.ed4c70.2
Index	8
TRef	V10
ERef	F9.71a09b.1

- **Hash signature in Ref enables retrieval of components**

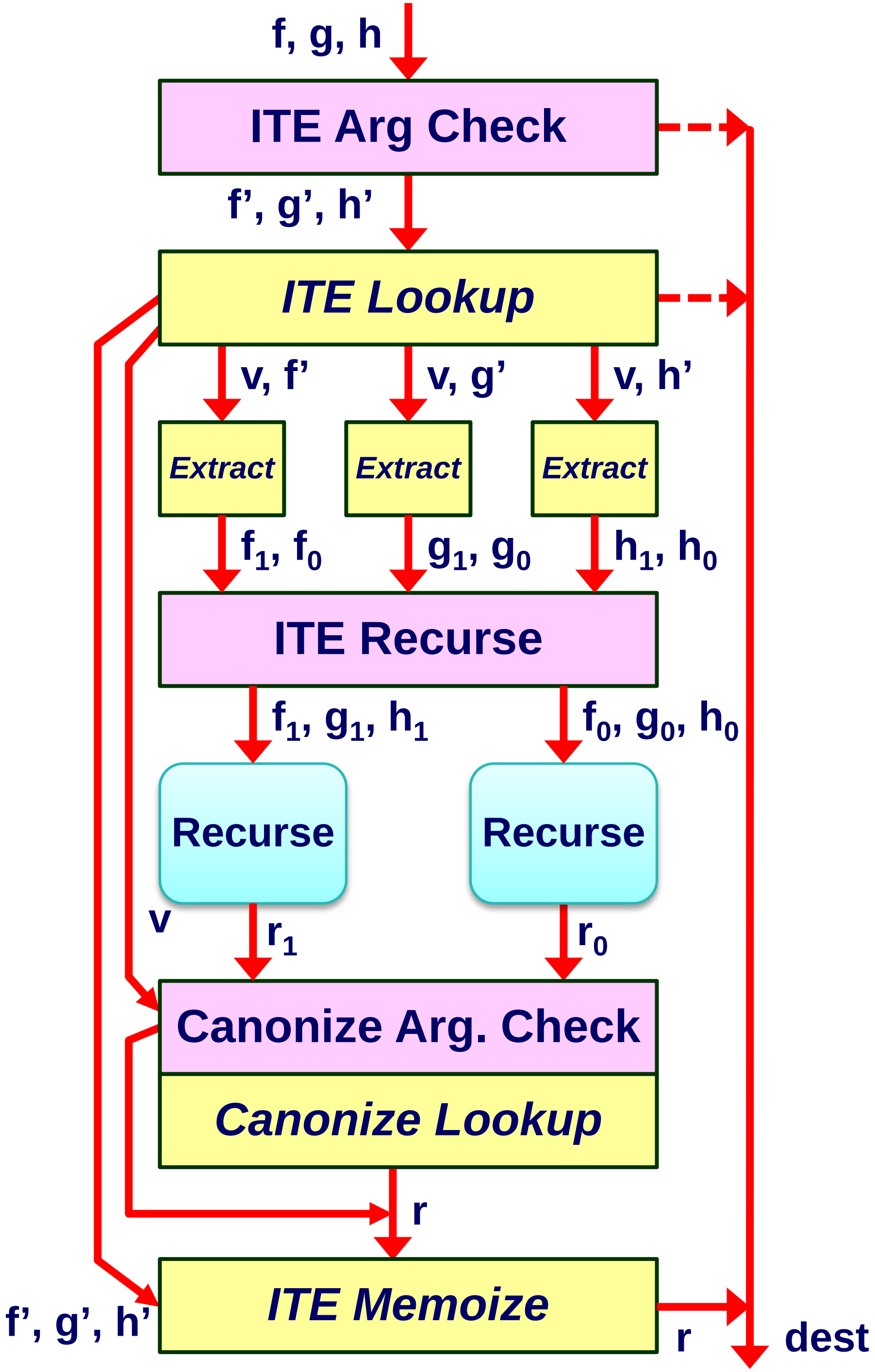
Data Flow Execution Model



Concept

- Computation expressed as dynamically generated network of *operators*
- Operator has fixed number of operands + destination
- When all operands available, operator *fires*
  - Perform computation
  - Send one or more operands to other operators
  - Generate one or more operators
  - Disappear

Implementing ITE



f, g, h

ITE Arg Check

f', g', h'

ITE Lookup

v, f'

v, g'

v, h'

Extract

Extract

Extract

f_1, f_0

g_1, g_0

h_1, h_0

ITE Recurse

f_1, g_1, h_1

f_0, g_0, h_0

Recurse

Recurse

v

r_1

r_0

Canonize Arg. Check

Canonize Lookup

r

f', g', h'

ITE Memoize

r

dest

🟨 Hashed placement

🟪 Arbitrary placement

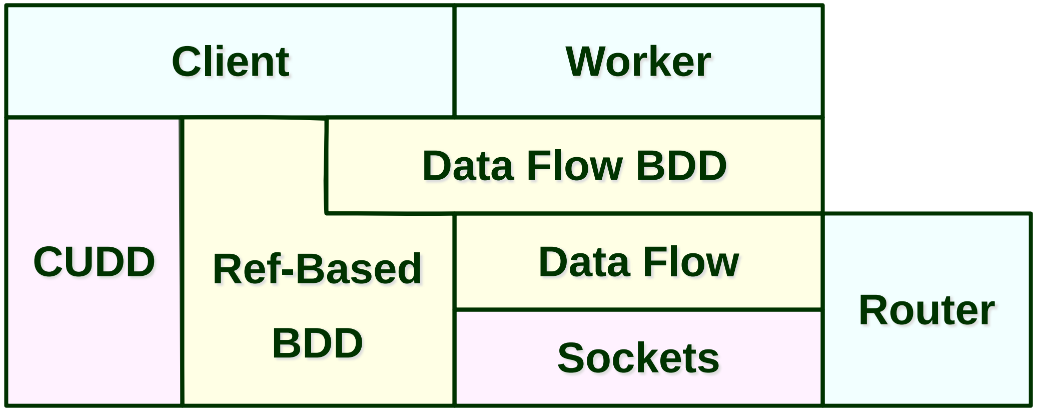
Request

- Compute $(f \wedge g) \vee (\neg f \wedge h)$
- Send result r to dest

Outcomes

- Early termination if special case or result found in memo table
- Otherwise, up to 9 operations + 2 recursive calls

Implementation



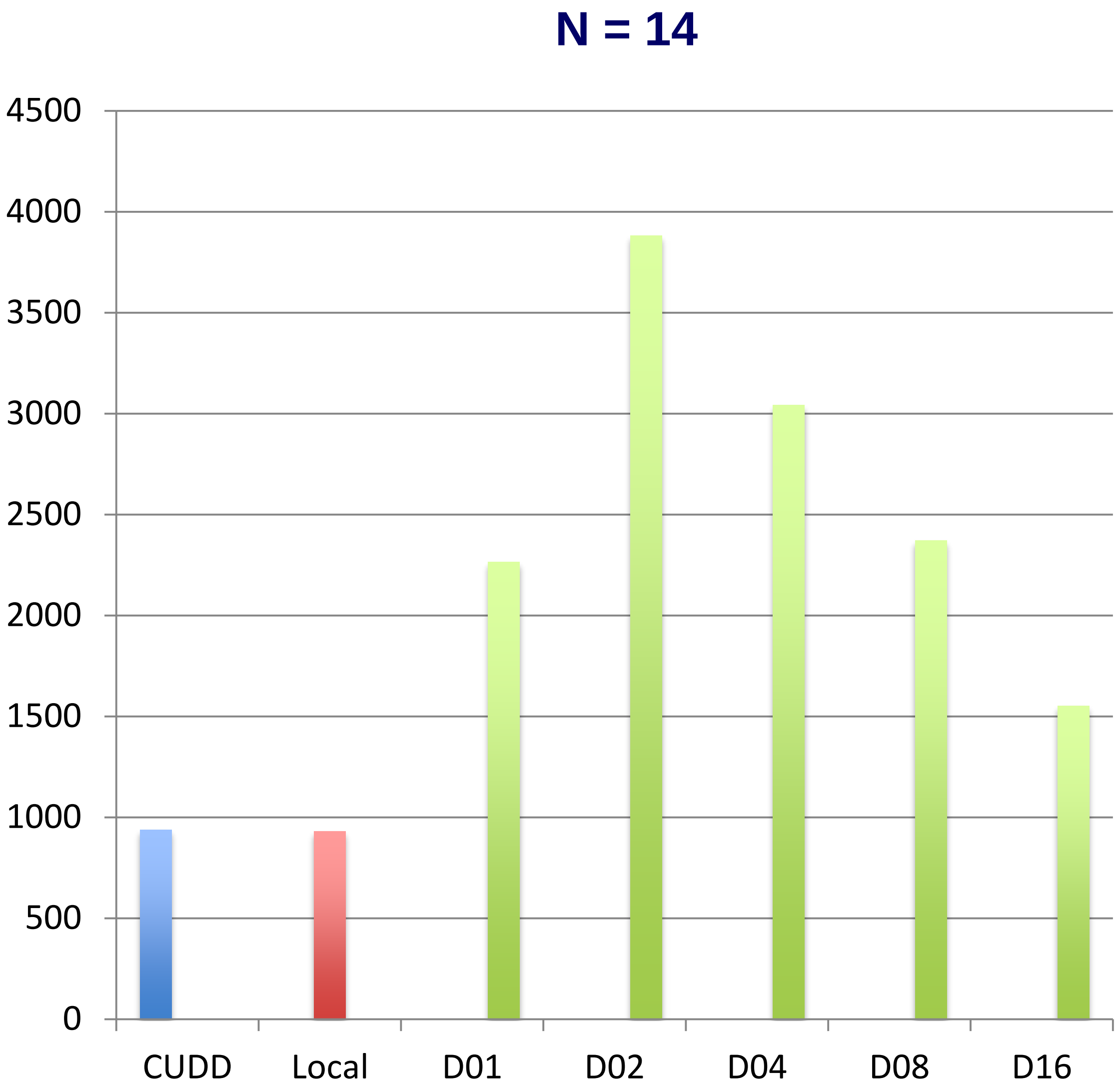
Data Flow BDD combination of:

- **General-purpose data flow on top of sockets interface**
- **Ref-based BDD**
  - Can also execute with standard, depth-first traversal

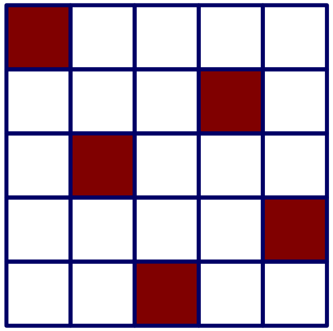
Client Interface

- **Any combination of data flow, sequential, CUDD**
  - Isomorphic results
  - For testing and performance comparison

Some Results

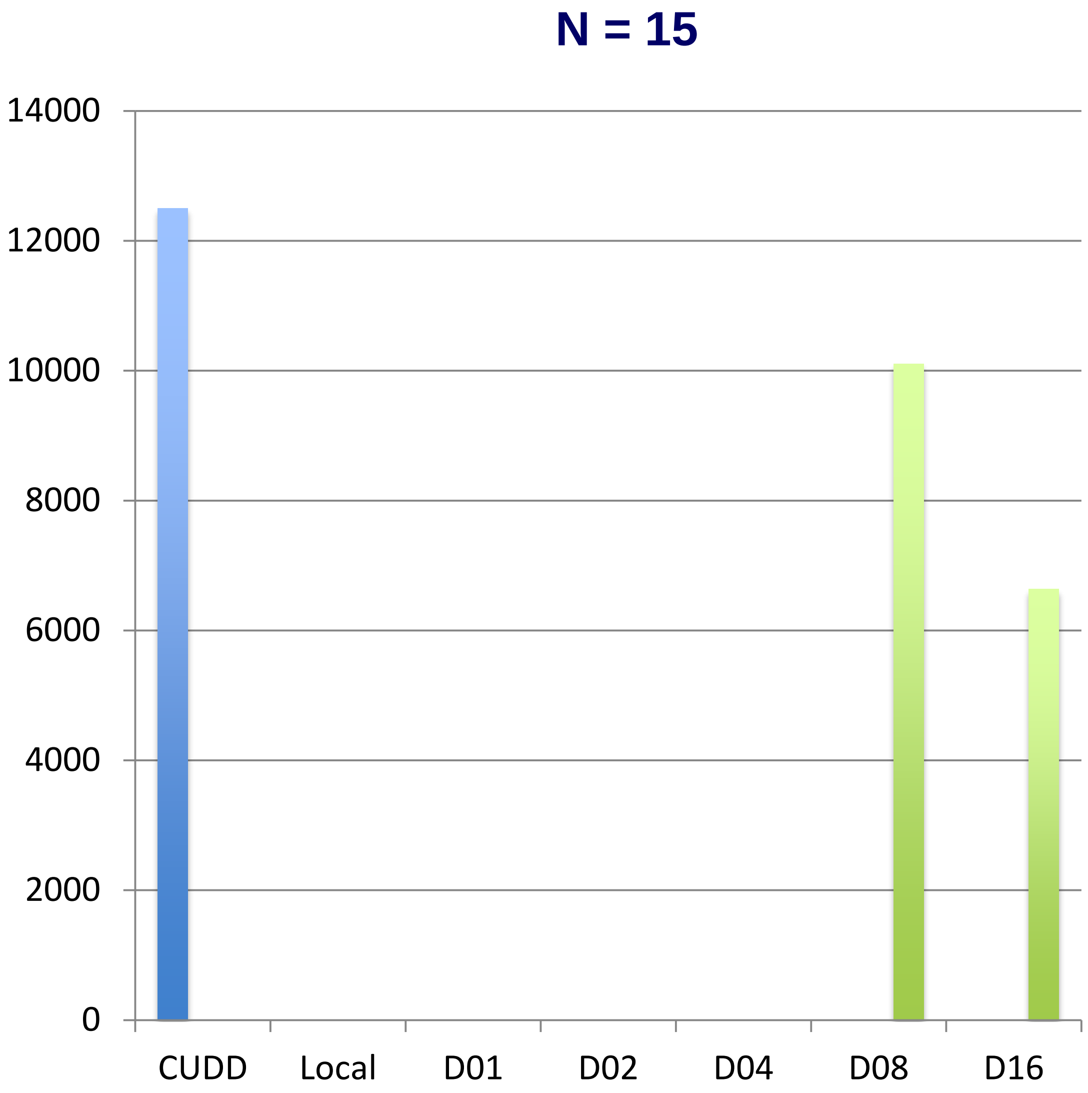


N-Queens Problem

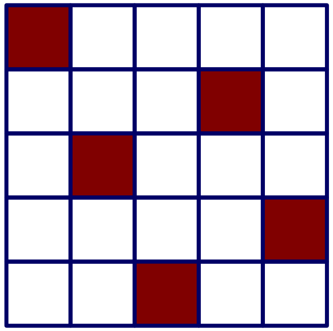


- With help from Hemanth Kini
- Boolean function representing all legal configurations
- Peak nodes = 23M
- Total ITEs = 233M
- Total OPs = 837M

More Results



N-Queens Problem



- Require 8 processors to have enough memory
- Peak nodes = 95M
- Total ITEs = 1.1B
- Total OPs = 3.9B

Implications

For BDDs

- **Scale to much larger sizes**
- **Allow sharing across multiple runs and users**
  - View as dynamically constructed, distributed database

For Parallel Computation

- **Execution model to support dynamic graph algorithms**
- **Combines data flow + distributed hash table**
  - Actions triggered by message passing
  - Locate objects by hash function
- **Features**
  - Overcome latency with high throughput
  - Scalable to arbitrary number of processors