
"Systemized" Program Analyses – A "Big Data" Perspective on Static Analysis Scalability

Harry Xu and Zhiqiang Zuo

University of California, Irvine

A Quick Survey

- Have you used a static program analysis?
What did you use it for?
- Have you designed a static program analysis?
- What are your major analysis infrastructures?
- Have you been bothered by its poor scalability?

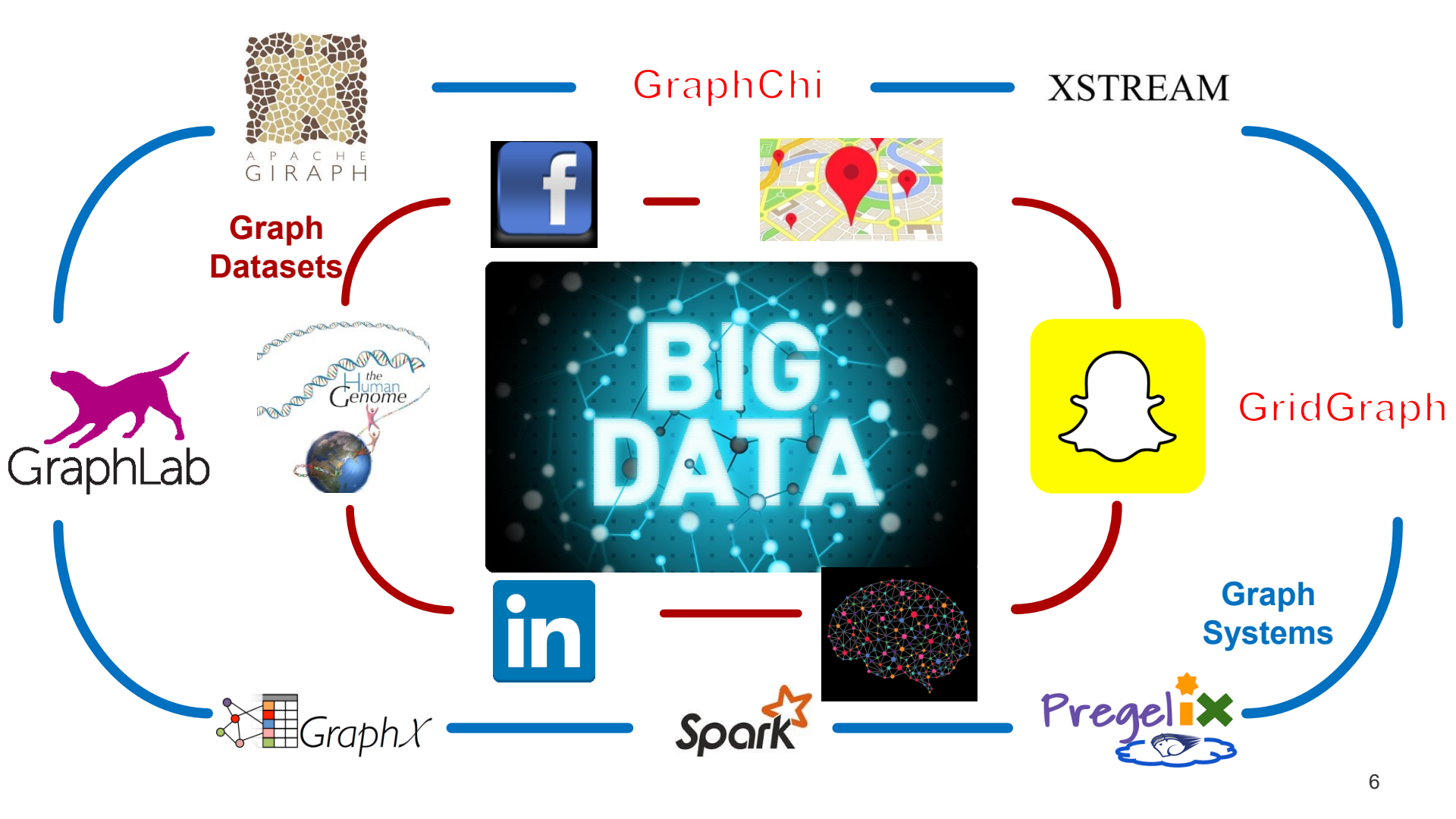
This Tutorial Is About

- What inspiration can we take from the big data community?
- How shall we shift our mindset from developing scalable analysis *algorithms* to developing scalable analysis *systems*?



Outline

- Background: big data/graph processing systems
- Treating static analysis as a big data problem
- Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
- BigSAT: distributed SAT solving at scale



Intimacy Between Systems and App. Areas



- **Machine Learning**
- **Information Retrieval**
- **Bioinformatics**
- **Sensor Networks**

.....

Large-Scale Graph Processing: Input

- Social network graphs
 - Twitter, Facebook, Friendster
- Bioinformatics graphs
 - Gene regulatory network (GRN)
- Map graphs
 - Google Map, Apple Map, Baidu Map
- Web graphs
 - Yahoo Webmap, UKDomain

Large-Scale Graph Processing: Input Size

- Social network graphs
 - Facebook: 721M vertices (users), 68.7B edges (friendships) in May 2011
- Map graphs
 - Google Map: 20 petabytes of data
- Web graphs
 - Yahoo Webmap: 1.4B websites (vertices) and 6.4B links (edges)

What Do These Numbers Mean

[To analyze the Facebook graph] calculations were performed on a Hadoop cluster with 2,250 machines, using the Hadoop/Hive data analysis framework developed at Facebook.

- Ugander et al., The Anatomy of the Facebook Social Graph, arXiv:1111.4503, 2011

Large-Scale Graph Processing: Core Idea

- Shift our mind from developing specialized graph algorithms to developing simple programs powered by large-scale systems
- Gather-apply-scatter: a graph-parallel abstraction

Think like a vertex

```
PageRank (Vertex v){  
  Gather {  
    foreach (e in v.inEdge) {  
      total += e.value;  
    }  
  }  
  Apply {  
    v.value = 0.15 * (0.85+total);  
  }  
  Scatter {  
    foreach (e in v.outEdge) {  
      e.value = v.value;  
    }  
  }  
}
```

Large-Scale Graph Processing: Classification I

- Distributed systems
 - GraphLab, PowerGraph, PowerLira, GraphX, Gemini
 - Challenges in communication reduction and partitioning
- Single machine systems
 - Shared memory: Ligra, Galois
 - Out of core: GraphChi, X-Stream, GridGraph, GraphQ
 - Challenges in disk I/O reduction

Large-Scale Graph Processing: Classification II

- Vertex-centricity
 - When computation is performed for a vertex, all its incoming/outgoing edges need to be available
 - GraphChi, PowerGraph, etc.
- Edge-centricity
 - Computation is divided into several phases
 - Vertex computation does not need all edges available
 - X-Stream, GridGraph, etc.

One Stone, Two Birds

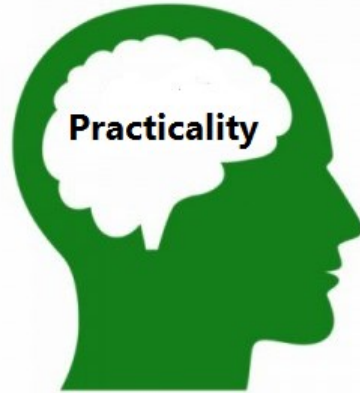
- Present a simple interface to the user, making it easy to develop graph algorithms
- Push performance optimizations down to the system, which leverages parallelism and various kind of support to improve performance and scalability

Outline

- Background: big data/graph processing systems
- Treating static analysis as a big data problem
- Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
- BigSAT: distributed SAT solving at scale

Where Is PL's Position in Big Data?

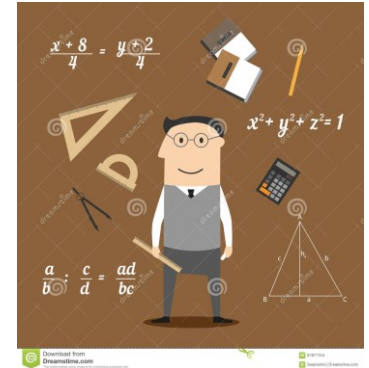
What Kind of Mindset Do You Have?



Systems

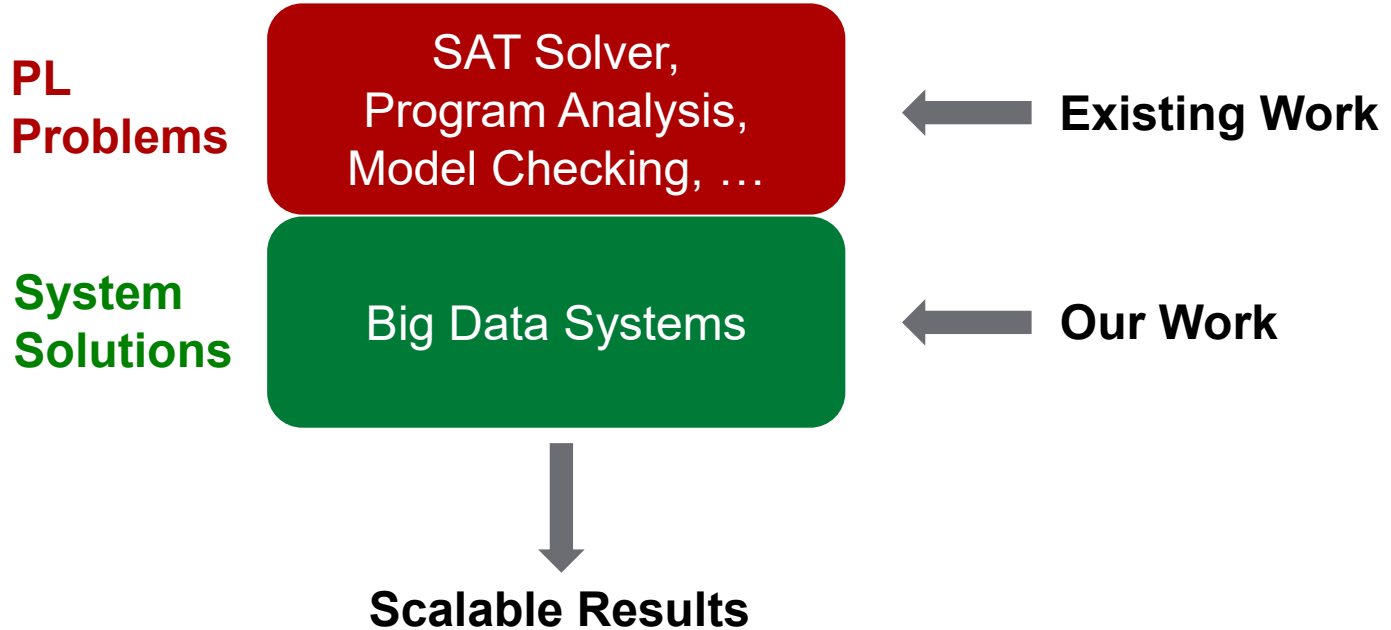


PL



Programming languages is a big source of data

PL Is Another Source of Big Data



Static Analysis Scalability Is A Big Concern

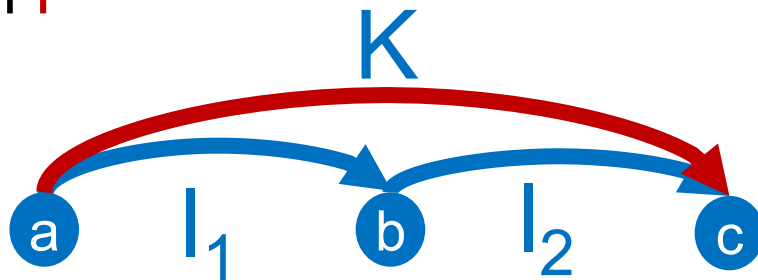
- An important PL problem: **Context-sensitive static analysis of very large codebases**

- ✧ Pointer/alias analysis
- ✧ Dataflow analysis
- ✧ May/must analysis
- ✧ ...

- ✧ Linux kernel
- ✧ Large server applications
- ✧ Distributed data-intensive systems
- ✧ ...

Context-Free Language (CFL) Reachability

- A program graph P



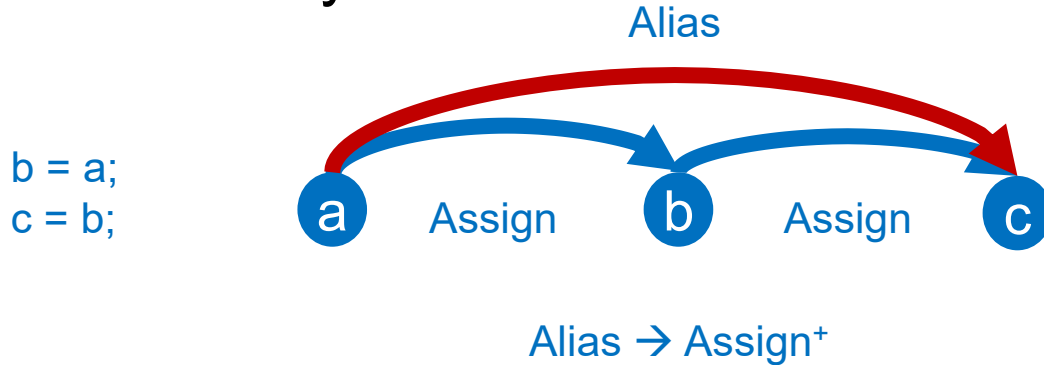
c is K -reachable from a

- A context-free Grammar G with balanced parentheses properties

$$K \rightarrow l_1 l_2$$

A Wide Range of Applications

- Pointer/alias analysis

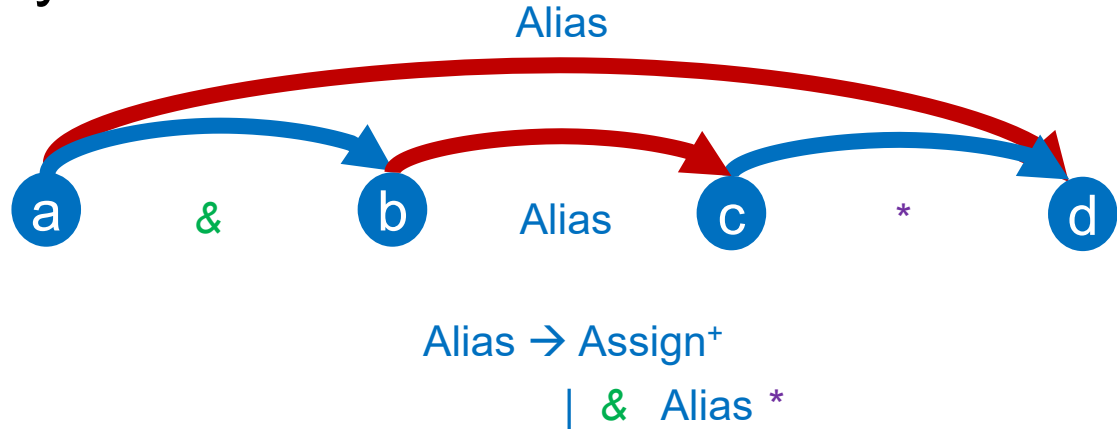


- Dataflow analysis, pushdown systems, set-constraint problems can all be converted to context-free-language reachability problems

A Wide Range of Applications (Cont.)

- Pointer/alias analysis

```
b = & a; // Address-of  
c = b;  
d = *c; // Dereference
```

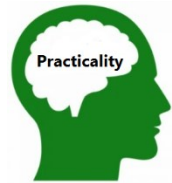


- *Address-of &* / *dereference** are the open/close parentheses



A Typical PL Problem

- **Traditional Approach**: a worklist-based algorithm
 - the worklist contains reachable vertices
 - no transitive edges are added physically
- **Problem**: embarrassingly sequential and unscalable
- **Solution**: develop approximations
- **Problem**: less precise and still unscalable



No Worry About Memory Blowup

- As long as one knows how to use disks and clusters

- Big Data thinking:

Solution =

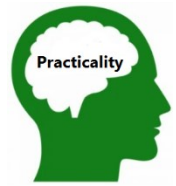
(1) Large Dataset + (2) Simple Computation +

System Design

Outline

- Background: big data/graph processing systems
- Treating static analysis as a big data problem
- **Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads**
- BigSAT: distributed SAT solving at scale

Turning Big Code Analysis into Big Data Analytics



- Key insights:
 - Adding transitive edges *explicitly* – satisfying (1)
 - Core computation is *adding edges* – satisfying (2)
 - Leveraging disk support for memory blowup
- Can existing graph systems be directly used?
 - No, none of them support dynamic addition of a lot of edges
 - (1) Online edge duplicate check and (2) dynamic graph repartitioning

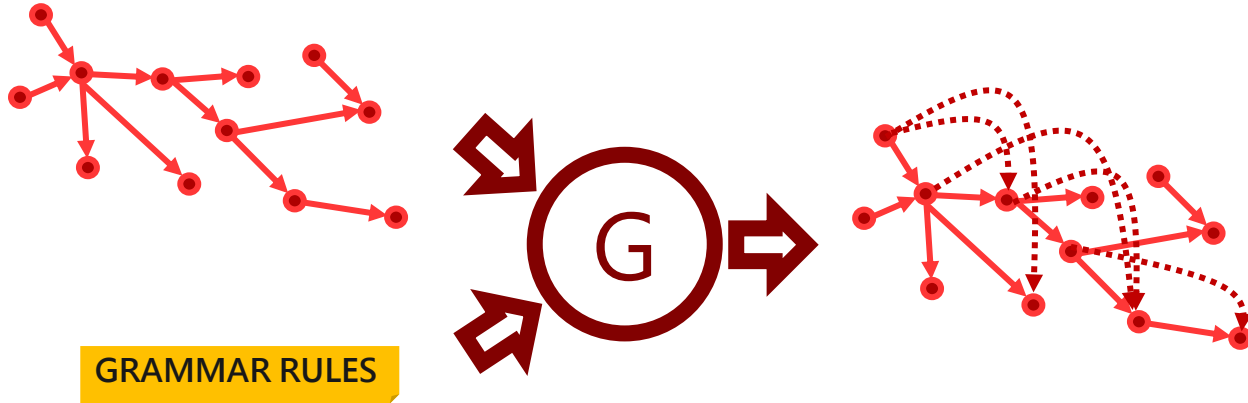
Graspan: A Graph System for Interprocedural Static Analysis of Large Programs

- Scalable
 - Disk-based processing on the developer's work machine
- Parallel
 - Edge-pair centric computation
- Easy to implement a static analysis
 - Developer only needs to generate graphs in mechanical ways and provide a context-free grammar to implement the analysis

4 students + 1 postdoc, 1.5 years of development; implemented in both Java and C++

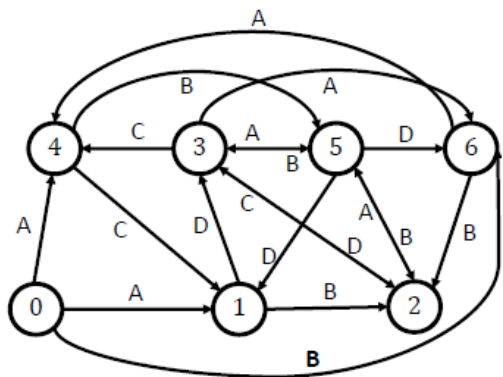
<https://github.com/Graspan/>

How It Works?



- Comparisons with a single-machine Datalog engine:
 - Grasp is a single-machine, out-of-core system
 - Grasp provides better locality and scheduling
 - Grasp is 3X faster than LogicBlox and 5X faster than Socialite even on small graphs

Granspan Design



Partition 0			Partition 1			Partition 2		
Src	Dst	Label	Src	Dst	Label	Src	Dst	Label
0	1	A	3	2	D	5	1	D
	4	A		4	C		2	B
1	2	B		5	B		3	A
	3	D		6	A		6	D
2	3	C	4	1	C	6	2	B
	5	A		5	B		4	A

- Partitions are of similar sizes
- Each partition contains an adjacency list of edges
- Edges in each partition are sorted

Preprocessing

Edge-Pair Centric
Computation

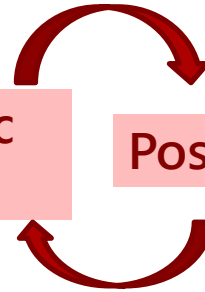
Post-Processing

Computation Occurs in Supersteps

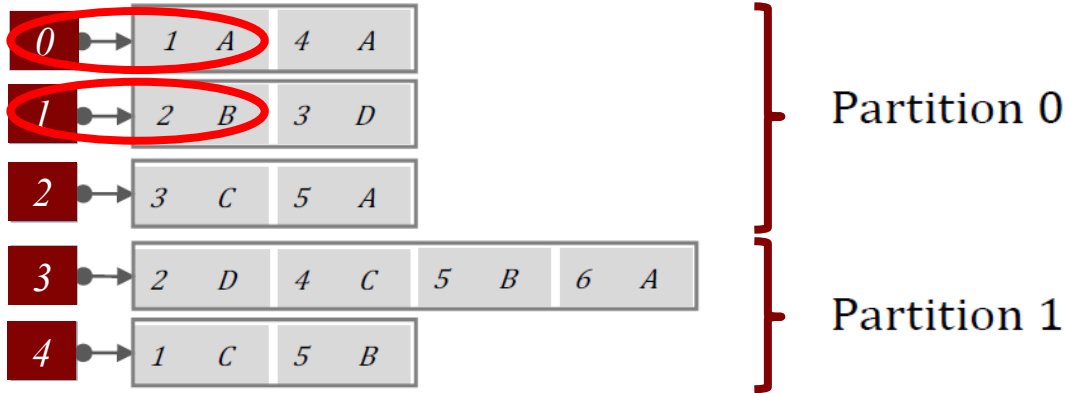
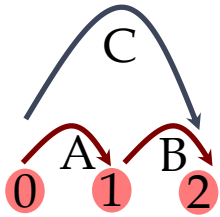
Preprocessing

Edge-Pair Centric
Computation

Post-Processing



Each Superstep Loads Two Partitions



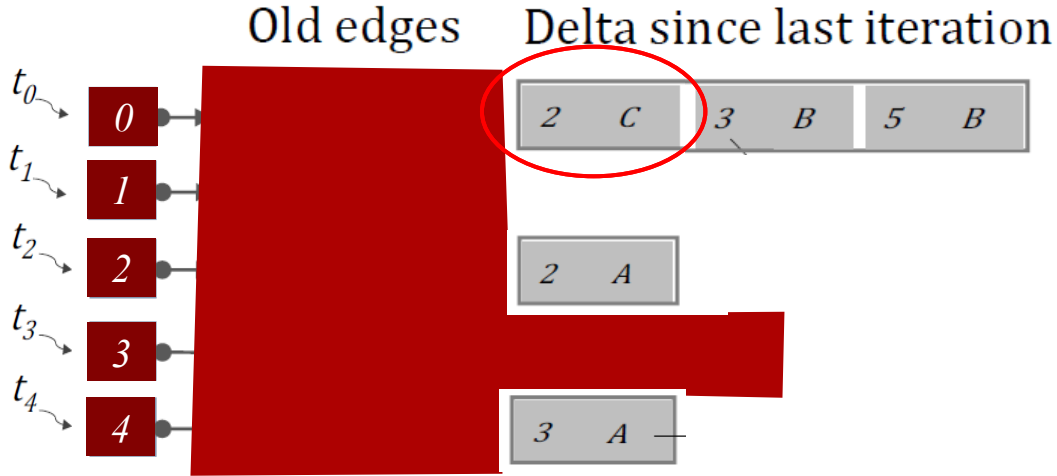
Grammar: $C := AB$ $D := BC$ $B := AD$ $A := CD$

Preprocessing

Edge-Pair Centric
Computation

Post-Processing

Each Superstep Loads Two Partitions



Grammar: $C = AB$, $D = BC$, $E = AD$, $A := CD$
 We keep iterating until delta is 0

Preprocessing

Edge-Pair Centric
Computation

Post-Processing

Post-Processing

- Repartition oversized partitions to maintain balanced load on memory
- Save partitions to disk
- Scheduler favors in-memory partitions and those with higher *matching degrees*

Preprocessing

Edge-Pair Centric
Computation

Post-Processing

What We Have Analyzed

Program	#LOC	#Inlines
Linux 4.4.0-rc5	16M	31.7M
PostgreSQL 8.3.9	700K	290K
Apache httpd 2.2.18	300K	58K

- With
 - A fully context-sensitive pointer/alias analysis
 - A fully context-sensitive dataflow analysis
- On a Dell Desktop Computer with 8GB memory and 1TB SSD

Evaluation Questions and Answers I

- Can the interprocedural analyses improve D. Englers' checkers?
 - Found 85 new NULL pointer bugs and 1127 unnecessary NULL tests in Linux 4.4.0-rc5

Checker	BL(4.4.0)		GR(4.4.0)		BL(2.6.1)
	RE	FP	RE	FP	RE
Block	0	0	0	0	43
Null	20	20	+108	23	98
Free	14	14	+4	4	21
Range	1	1	0	0	11
Lock	15	15	+3	3	5
Size	25	23	+11	11	3
UNTest	N/A	N/A	+1127	0	N/A

Evaluation Questions and Answers II

- Sample bugs

```
void*probe_kthread_data(
    task_struct *task){
    void *data = NULL;
    probe_kernel_read(&data);

    /*data will be
    dereferenced after
    return.*/
    return data;
}

long probe_kernel_read
(void *dst){
    if(...)
        return -EFAULT;
    return
    __probe_kernel_read(dst);
}
```

(a) NULL deref in kernel/kthread.c

```
#define page_private(page)
    ((page)->private)

bool swap_count_continued
(...){
    head=vmalloc_to_page(...);
    if(page_private(head)
        != ...){
        ...
    }
}

page*vmalloc_to_page(...){
    page *page = NULL;
    if (!pgd_none(*pgd)){
        //...
    }
    return page;
}
```

(b) NULL deref in mm/swapfile.c

Evaluation Questions and Answers III

- Bug breakdown in modules

Modules	NULL pointer defs	Unnecessary NULL Tests
arch	0	75
crypto	0	15
init	0	1
kernel	4 (2)	65
mm	3 (0)	84
security	0	78
block	6 (2)	31
fs	19 (3)	84
ipc	0	17
lib	0	39
net	10 (8)	269
sound	15 (5)	83
drivers	25 (3)	286
Total	108 (23)	1127

Evaluation Questions and Answers IV

- Is Graspan efficient and scalable?
 - Computations took 11 mins – 12 hrs

Prog	Pointer/Alias Analysis				
	IS=(E,V)	PS=(E,V)	PT	SS	T
Linux	(249.5M,52.9M)	(1.1B,52.9M)	91 secs	27	1.7 hrs
PSQL	(25.0M,5.2M)	(862.2M,5.2M)	10 secs	16	6.0 hrs
httpd	(8.2M, 1.7M)	(904.3M, 1.7M)	3 secs	13	8.4 hrs

Prog	Dataflow Analysis				
	IS=(E,V)	PS=(E,V)	PT	SS	T
Linux	(69.4M, 63.0M)	(211.3M, 63.0M)	65 secs	33	11.9 hrs
PSQL	(34.8M,29.0M)	(56.1M, 29.0M)	35 secs	16	2.4 hrs
httpd	(10.0M, 5.3M)	(19.3M, 5.3M)	9 secs	16	11.4 mins

Evaluation Questions and Answers V

- Graspan v/s other engines?
 - GraphChi crashed in 133 secs

Analysis	Graspan		ODA [101]	Socialite [45]
	CT	I/O		
Linux-P	99.7 mins	46.6 secs	OOM	OOM
Linux-D	713.8 mins	4.2 mins	-	OOM
PostgreSQL-P	353.1 mins	4.5 mins	> 1 day	OOM
PostgreSQL-D	143.8 mins	57.1 secs	-	OOM
httpd-P	497.9 mins	7.6 mins	> 1 day	> 1 day
httpd-D	11.3 mins	3.3 secs	-	4 hrs

[101] X. Zheng and R. Rugina, Demand-driven alias analysis for C, POPL, 2008

[45] M. S. Lam, S. Guo, and J. Seo. Socialite: Datalog extensions for efficient social network analysis. ICDE, 2013.

Evaluation Questions and Answers VI

- How easy to use Graspan?
 - 1K LOC of C++ for writing each of points-to and dataflow graph generators
 - Provide a grammar file
- Data structure analysis in LLVM
 - More than 10K lines of code

Download and Use Graspan

- <https://github.com/Graspan>
- Two versions available at GitHub
 - <https://github.com/Graspan/graspan-cpp>
 - <https://github.com/Graspan/graspan-java>
- Data structure analysis in LLVM
 - More than 10K lines of code

Outline

- Background: big data/graph processing systems
- Treating static analysis as a big data problem
- Graspan: an out-of-core graph system for parallelizing and scaling static analysis workloads
- **BigSAT: distributed SAT solving at scale**

Outline

- Preliminaries
- DPLL & CDCL
- Parallelizability of SAT solving
- BigSAT

Boolean Satisfiability Problem (SAT)

- A propositional formula is built from propositional variables, operators (and, or, negation) and parentheses.

$$(x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4)$$

- SAT problem
 - Given a formula, find a satisfying assignment or prove that none exists.

CNF formula

$$(x_1' \vee x_2') \wedge (x_1' \vee x_2 \vee x_3') \wedge (x_1' \vee x_3 \vee x_4') \wedge (x_1 \vee x_4)$$

- **Literal**: a variable or negation of a variable
- **Clause**: a disjunction of literals
- **CNF**: a conjunction of clauses

Why is SAT important?

- Theoretically,
 - First NP-completeness problem [Cook,1971]
- Practically,
 - Hardware/software verification
 - Model checking
 - Cryptography
 - Computational biology
 - ...

DPLL

- Backtrack search
- Boolean constraint propagation (BCP)

$$(x_1') \wedge (x_1 \vee x_2) \wedge (x_2' \vee x_3')$$

DPLL

- Backtrack search
- Boolean constraint propagation (BCP)

$$(x1') \wedge (x1 \vee x2) \wedge (x2' \vee x3') \Rightarrow x1 = F$$

DPLL

- Backtrack search
- Boolean constraint propagation (BCP)

$$(x1') \wedge (x1 \vee x2) \wedge (x2' \vee x3') \Rightarrow x1=F \ x2=T$$

DPLL

- Backtrack search
- Boolean constraint propagation (BCP)

$$(x1') \wedge (x1 \vee x2) \wedge (x2' \vee x3') \Rightarrow x1=F \ x2=T$$

DPLL

- Backtrack search
- Boolean constraint propagation (BCP)

$$(x1') \wedge (x1 \vee x2) \wedge (x2' \vee x3') \Rightarrow x1=F \ x2=T \ x3=F$$

- Algorithm
 - Select a variable and assign T or F
 - Apply BCP
 - If there's a conflict, backtrack to previous decision level
 - Otherwise, continue until all variables are assigned

$$x_1 + x_4$$

$$x_1 + x_3' + x_8'$$

$$x_1 + x_8 + x_{12}$$

$$x_2 + x_{11}$$

$$x_7' + x_3' + x_9$$

$$x_7' + x_8 + x_9'$$

$$x_7 + x_8 + x_{10}'$$

$$x_7 + x_{10} + x_{12}'$$

$$x1 + x4$$

$$x1 + x3' + x8'$$

$$x1 + x8 + x12$$

$$x2 + x11$$

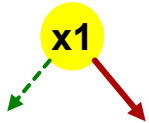
$$x7' + x3' + x9$$

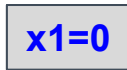
$$x7' + x8 + x9'$$

$$x7 + x8 + x10'$$

$$x7 + x10 + x12'$$


$$x1=0$$




$$x1=0$$

$$x1 + x4$$

$$x1 + x3' + x8'$$

$$x1 + x8 + x12$$

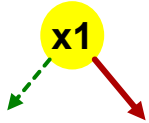
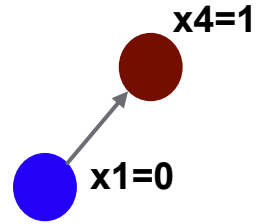
$$x2 + x11$$

$$x7' + x3' + x9$$

$$x7' + x8 + x9'$$

$$x7 + x8 + x10'$$

$$x7 + x10 + x12'$$



$$x1=0, x4=1$$

$$x1 + x4$$

$$x1 + x3' + x8'$$

$$x1 + x8 + x12$$

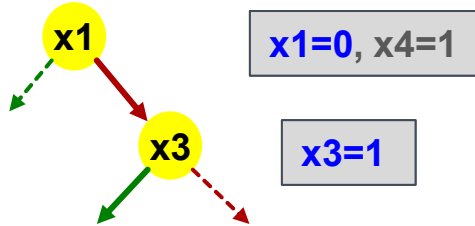
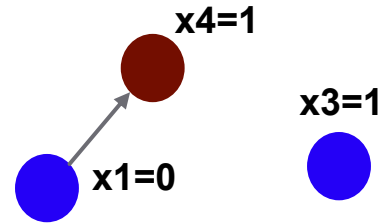
$$x2 + x11$$

$$x7' + x3' + x9$$

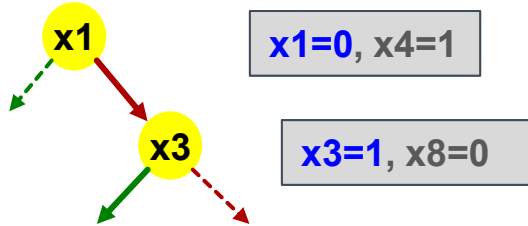
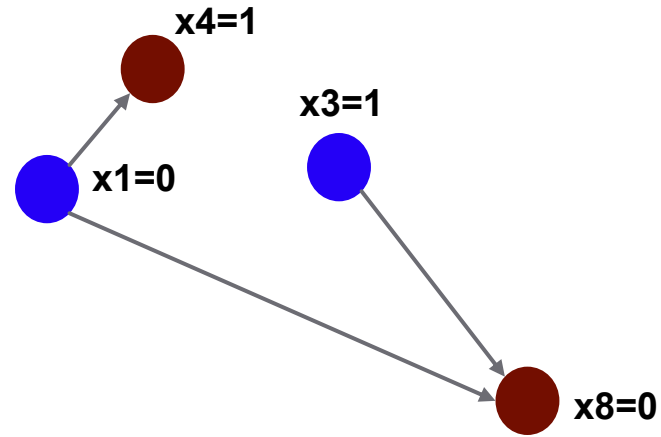
$$x7' + x8 + x9'$$

$$x7 + x8 + x10'$$

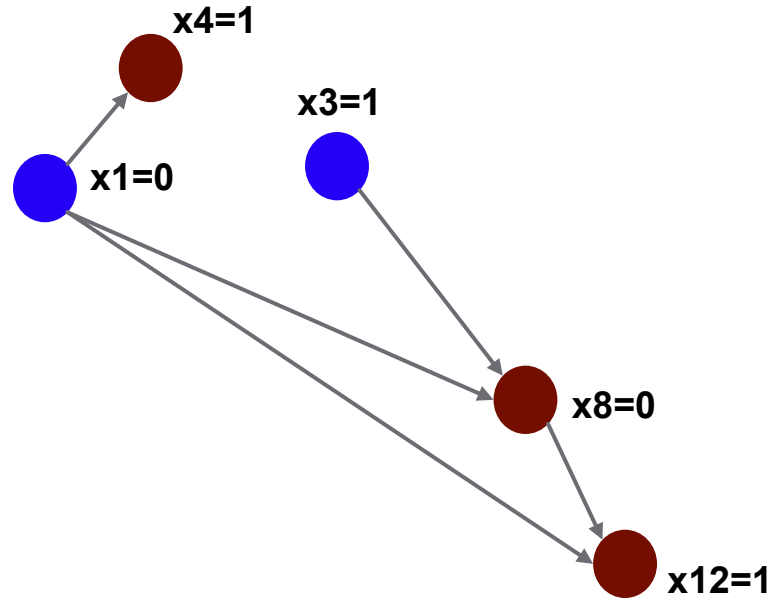
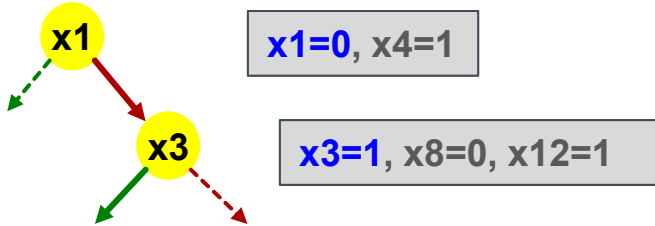
$$x7 + x10 + x12'$$



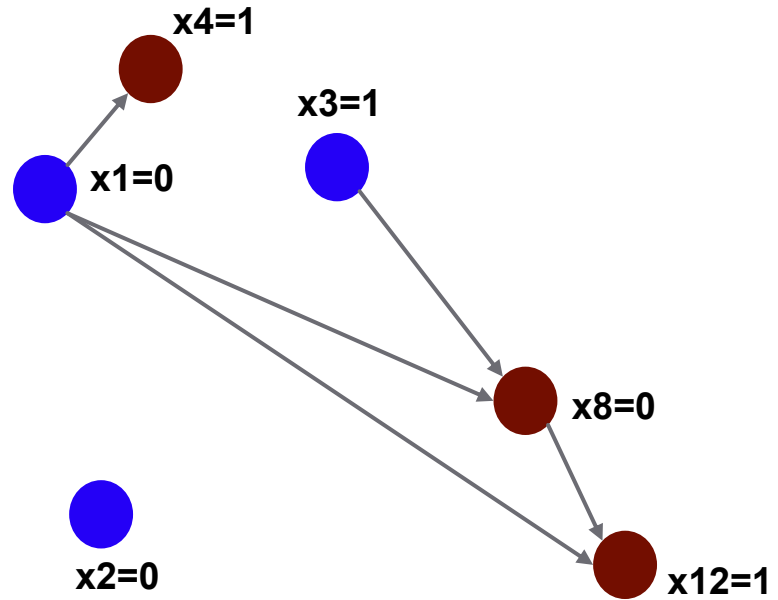
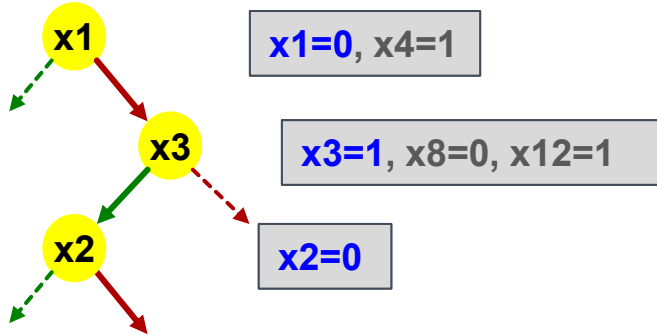
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



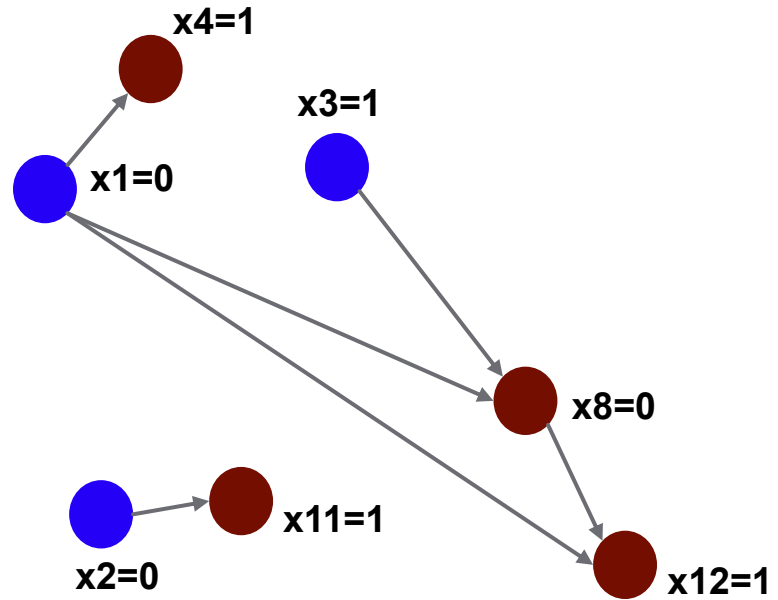
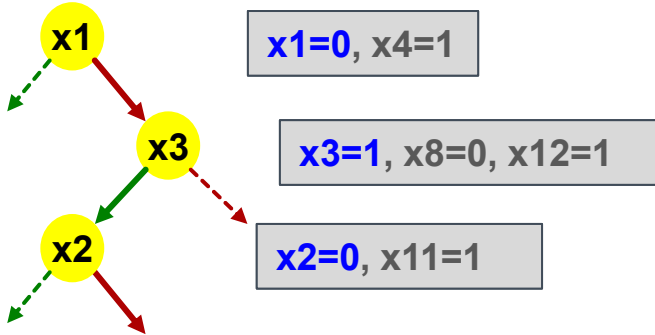
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



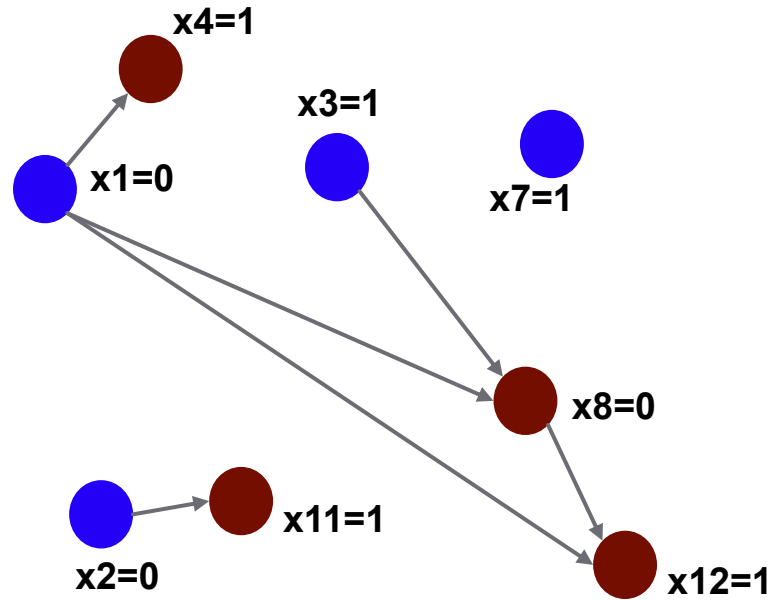
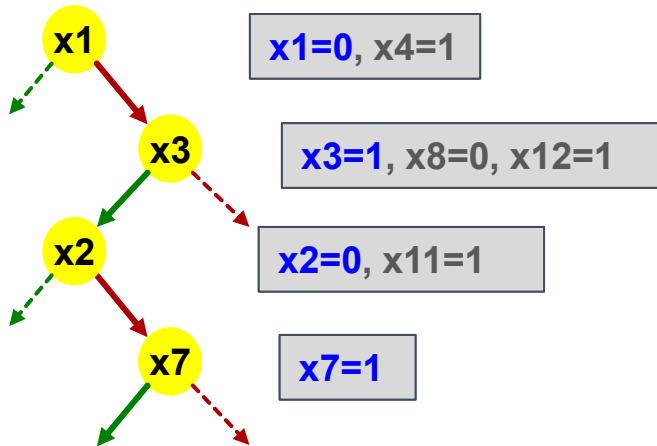
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



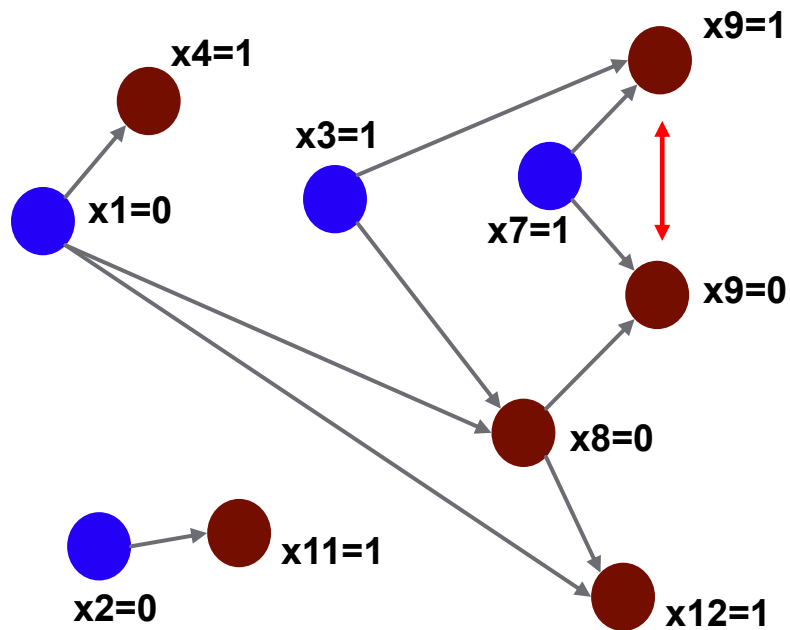
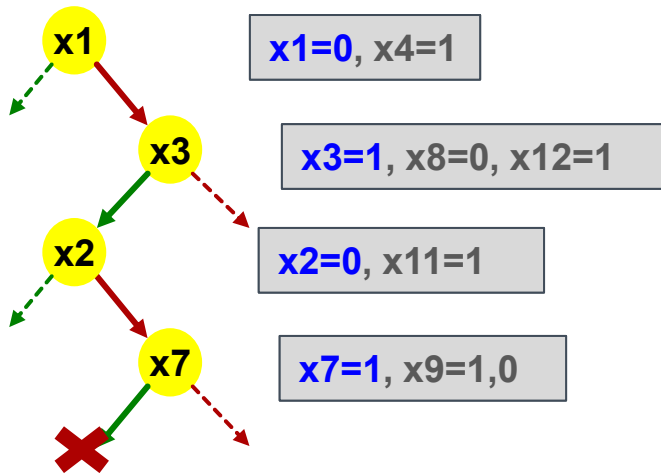
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



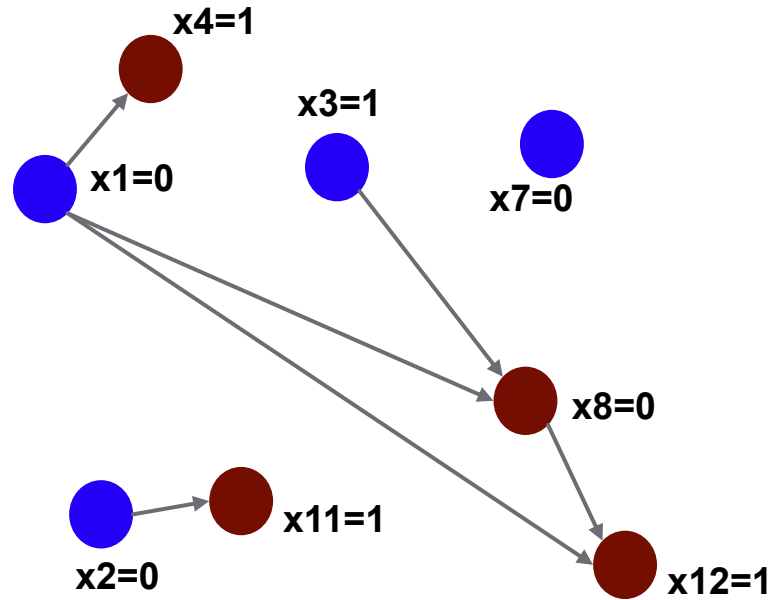
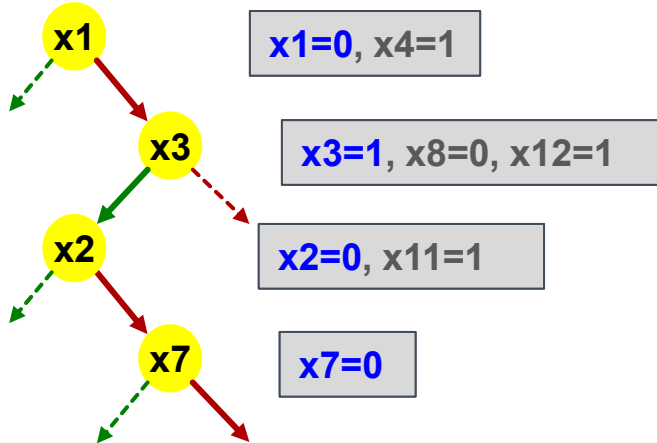
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



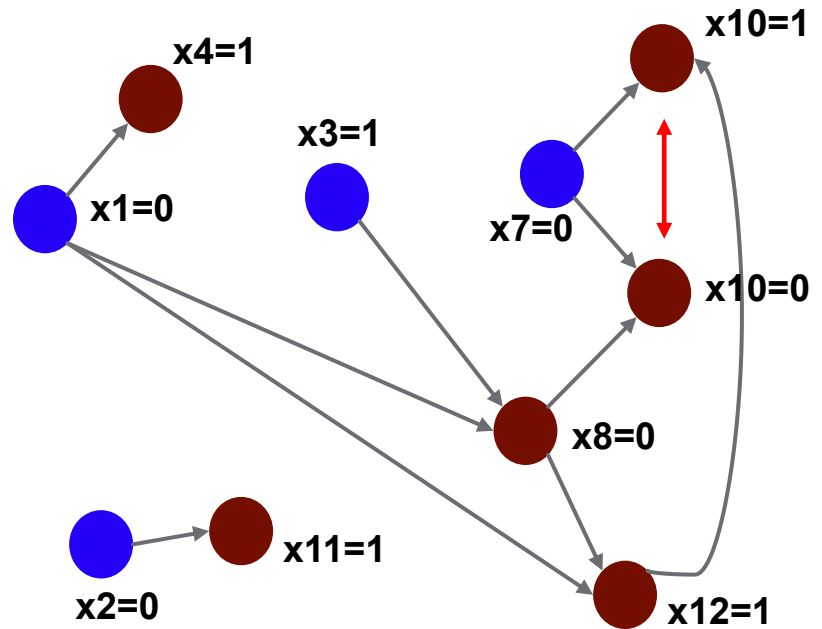
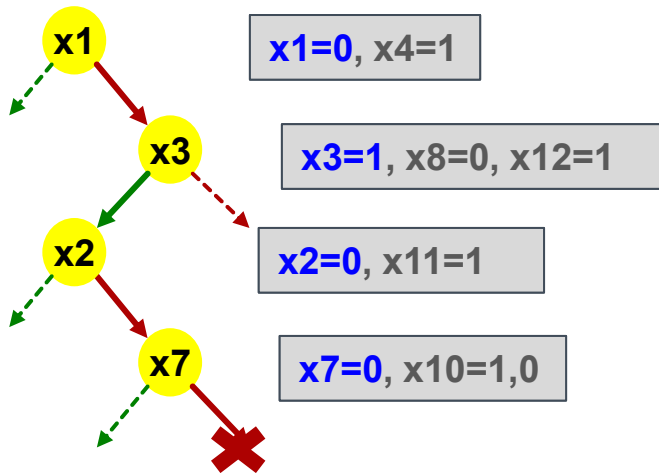
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



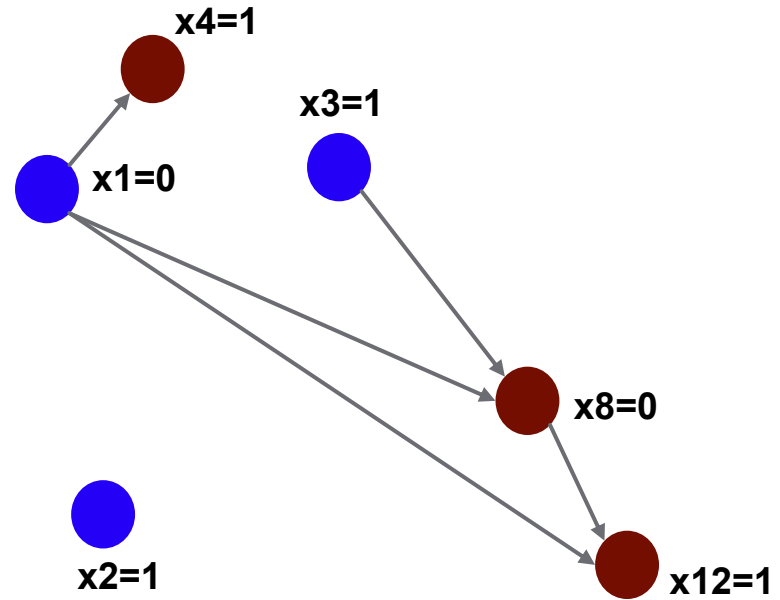
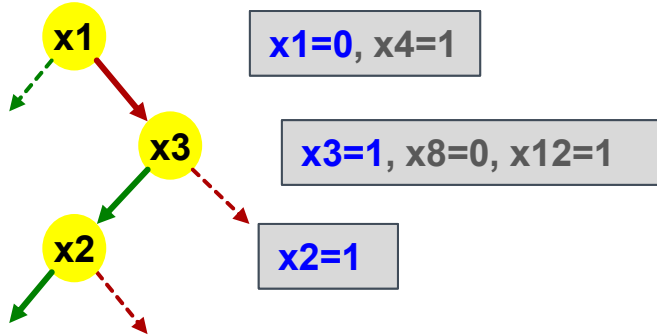
$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_{7'} + x_3' + x_9$
 $x_{7'} + x_8 + x_{9'}$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$



- $x1 + x4$
- $x1 + x3' + x8'$
- $x1 + x8 + x12$
- $x2 + x11$
- $x7' + x3' + x9$
- $x7' + x8 + x9'$
- $x7 + x8 + x10'$
- $x7 + x10 + x12'$



$x_1 + x_4$
 $x_1 + x_3' + x_8'$
 $x_1 + x_8 + x_{12}$
 $x_2 + x_{11}$
 $x_7' + x_3' + x_9$
 $x_7' + x_8 + x_9'$
 $x_7 + x_8 + x_{10}'$
 $x_7 + x_{10} + x_{12}'$

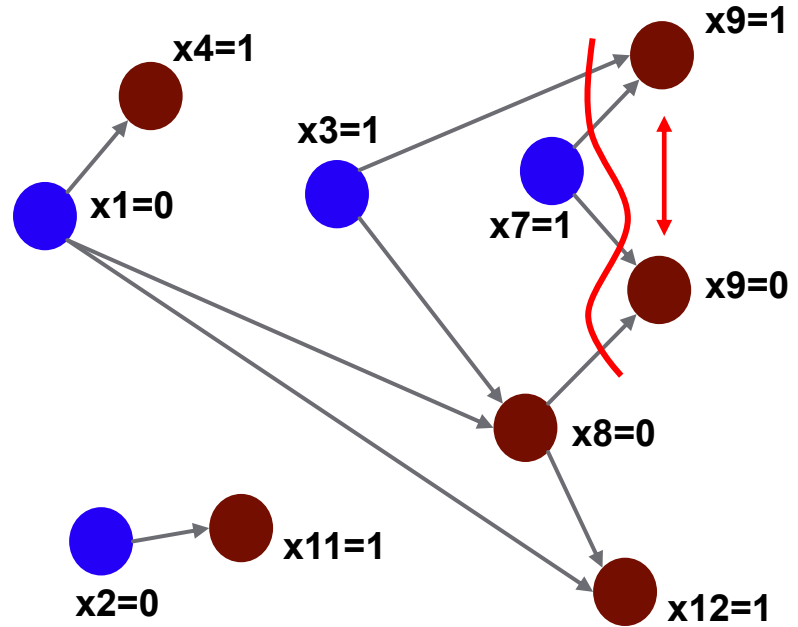
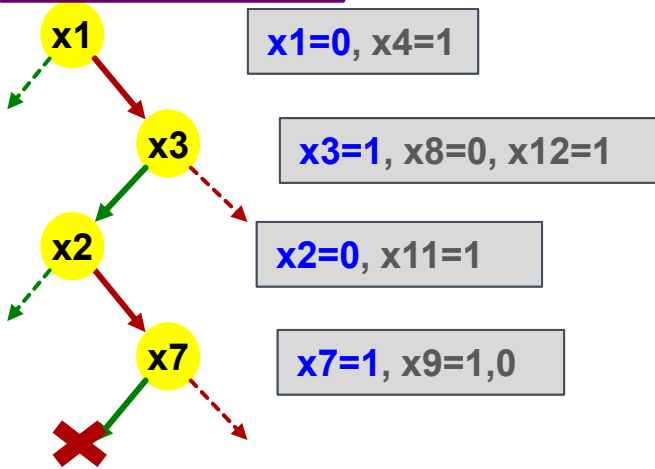


Conflict-driven clause learning (CDCL)

- Clause learning from conflicts
- Non-chronological backtracking
- Algorithm
 - Select a variable and assign T or F
 - Apply BCP
 - If there's a conflict, conflict analysis to learn clauses and backtrack to the appropriate decision level
 - Otherwise, continue until all variables are assigned

$x1 + x4$
 $x1 + x3' + x8'$
 $x1 + x8 + x12$
 $x2 + x11$
 $x7' + x3' + x9$
 $x7' + x8 + x9'$
 $x7 + x8 + x10'$
 $x7 + x10 + x12'$

$x3' + x7' + x8$

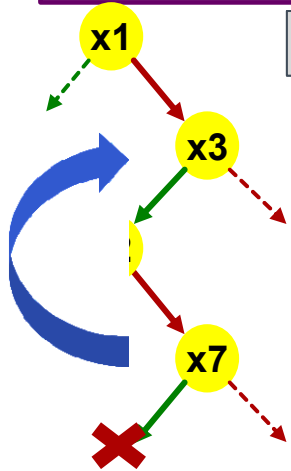


$x3=1 \wedge x7=1 \wedge x8=0 \longrightarrow \text{conflict}$

$(x3=1 \wedge x7=1 \wedge x8=0)'$

$x3' + x7' + x8$

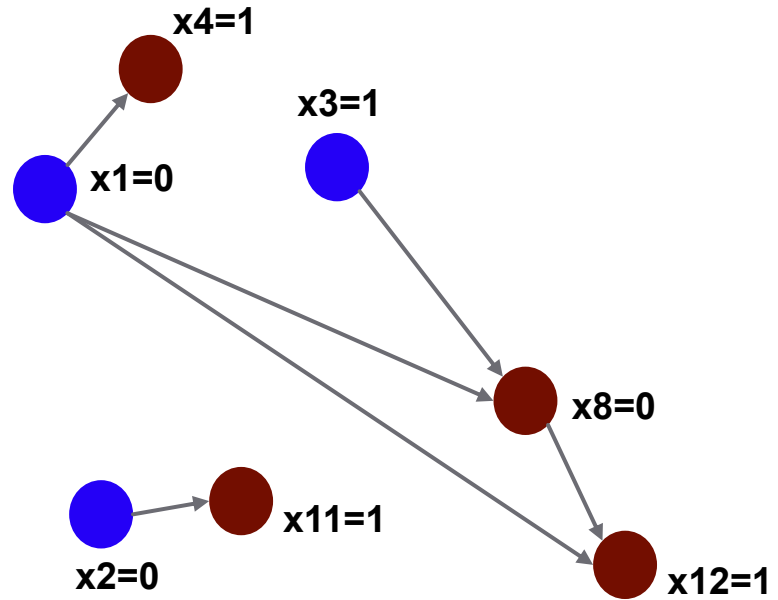
- $x_1 + x_4$
- $x_1 + x_3' + x_8'$
- $x_1 + x_8 + x_{12}$
- $x_2 + x_{11}$
- $x_7' + x_3' + x_9$
- $x_7' + x_8 + x_9'$
- $x_7 + x_8 + x_{10}'$
- $x_7 + x_{10} + x_{12}'$
- $x_3' + x_7' + x_8$**



$x_1=0, x_4=1$

$x_3=1, x_8=0, x_{12}=1$

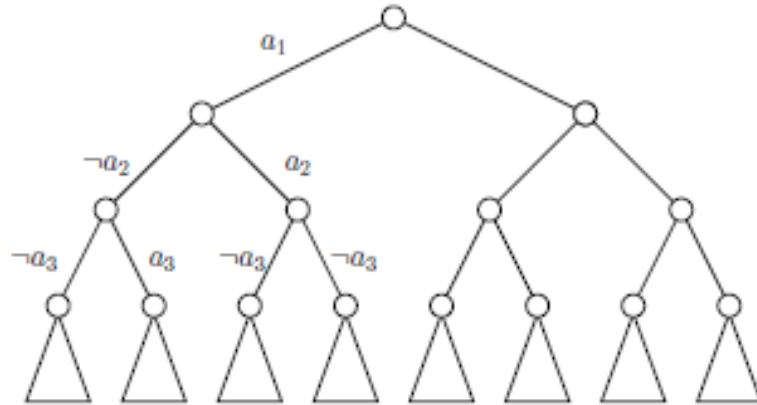
Backtrack to the decision level $x_3=1$



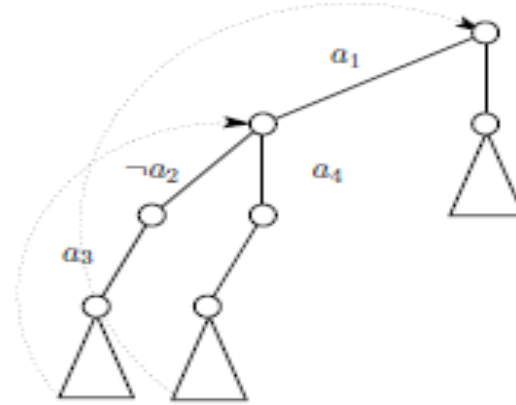
Conflict-driven clause learning (CDCL)

- Clause learning from conflicts
- Non-chronological backtracking
- Others
 - Lazy data structures
 - Branching heuristics
 - Restarts
 - Clause deletion
 - etc.

DPLL vs. CDCL



DPLL: no learning and chronological backtracking



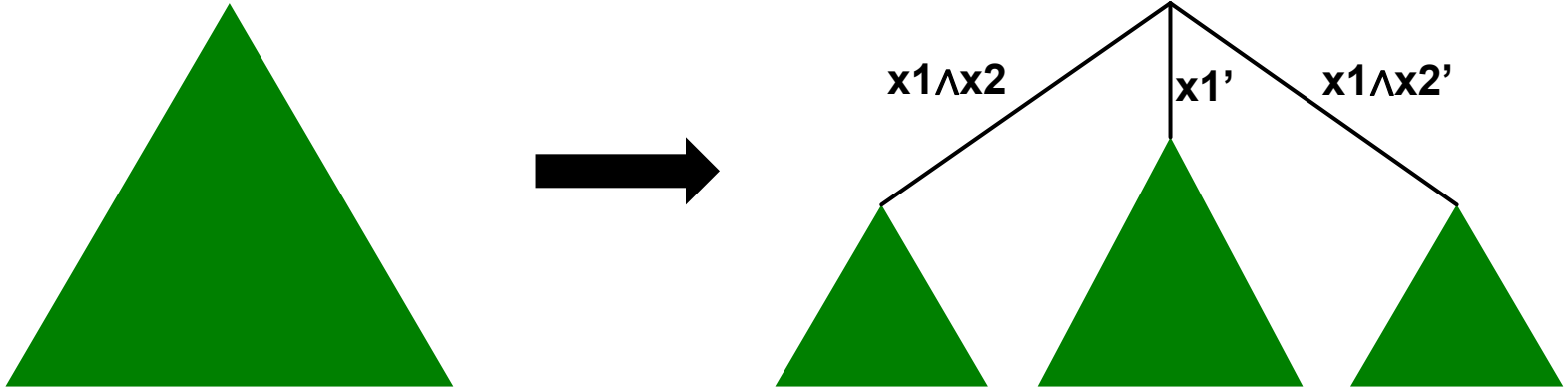
CDCL: clause learning and non-chronological backtracking

Parallel SAT solvers

- Why?
 - Sequential solvers are difficult to improve
 - Can't scale to large problems
- Category
 - Divide-and-conquer
 - Portfolio-based

Divide-and-conquer

- Divide search space into multiple independent sub-trees via guiding-paths

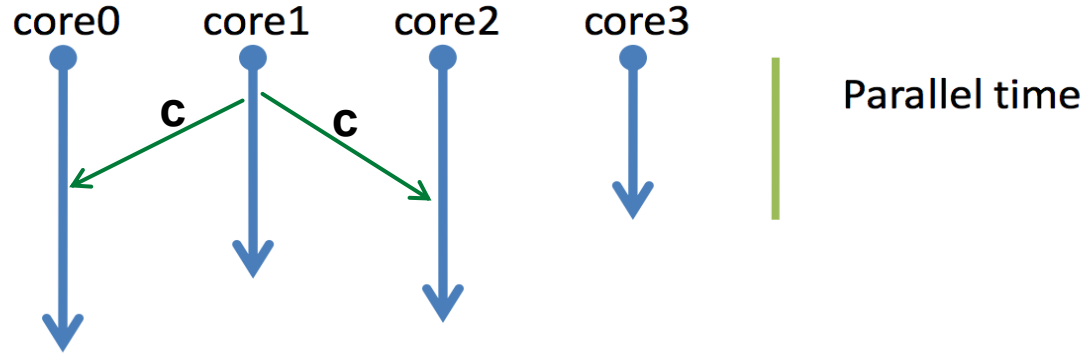


- Problem: **load imbalance**

Portfolio-based

- Observations
 - Modern SAT solvers are sensitive to parameters
- Principle
 - Run multiple CDCLs with different parameters simultaneously
 - Let them compete and cooperate

Portfolio-based



- Diversification
 - Restart, variable heuristics, polarity, learning scheme
- Clause sharing

Parallelization Barriers

- Poor scalability
 - 3x faster on 32-cores
- Reasons
 - BCP is P-complete, hard to parallelize
 - Bottlenecks [AAAI'2013]
 - Load imbalance for divide & conquer
 - Diversity for portfolio-based

Bottlenecks in CDCL proofs

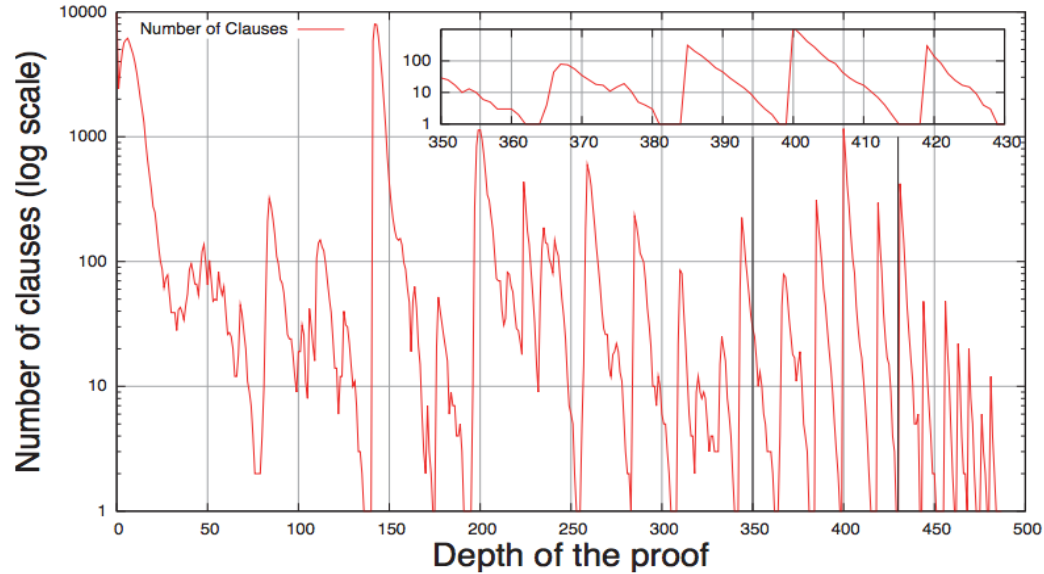
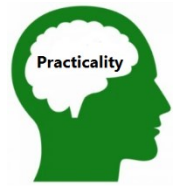


Figure 1: Number of clauses derived at each depth of a typical CDCL proof

BigSAT: Turning SAT (DP) into Big Data Analytics



- Big Data thinking:

Big Data Solution

||

(1) Large Dataset + (2) Simple Computation + **System Design**

- DPLL?
- Others?

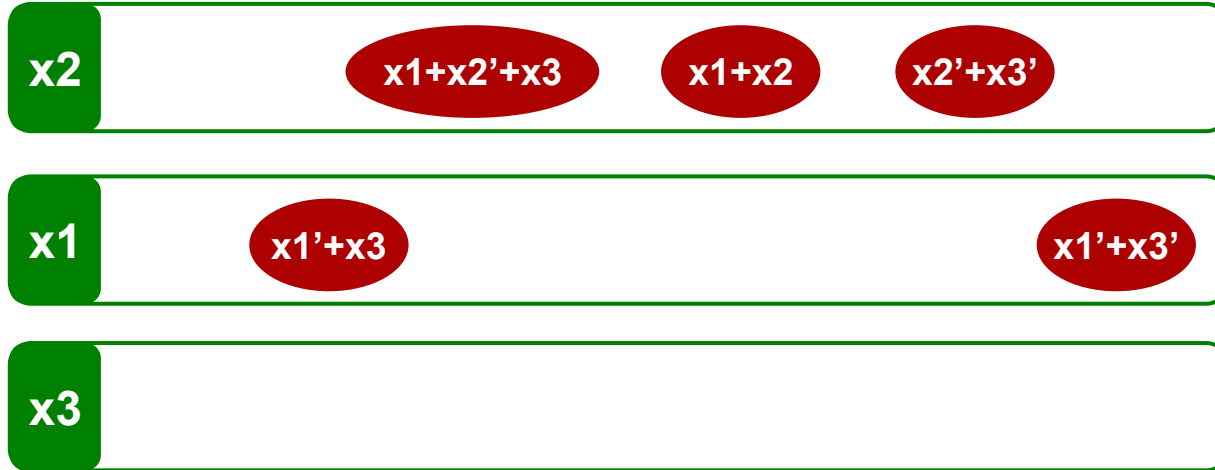
DP

- Introduced by Davis and Putnam in 1960
- Resolution

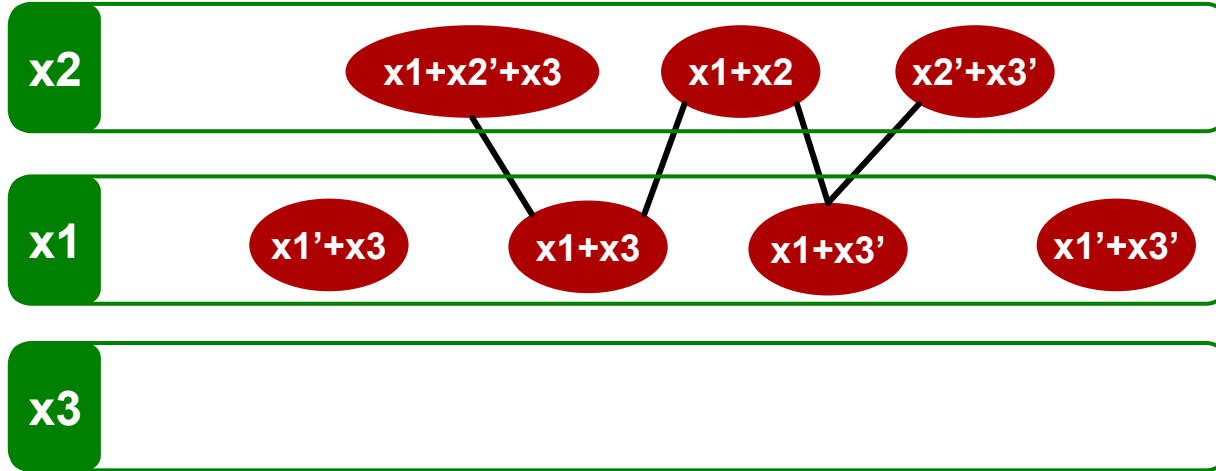
$$\frac{(x \vee y) \wedge (x' \vee z)}{(y \vee z)}$$

- Algorithm
 - Select a variable x , and add all resolvents
 - Remove all clauses containing x
 - Continue until no variable left for resolution

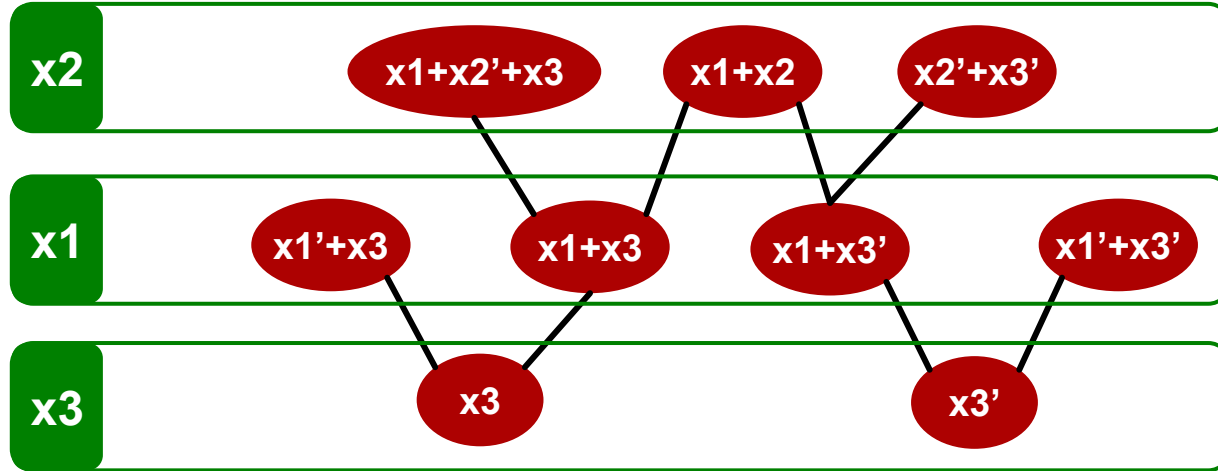
Ordering: $x_2 > x_1 > x_3$



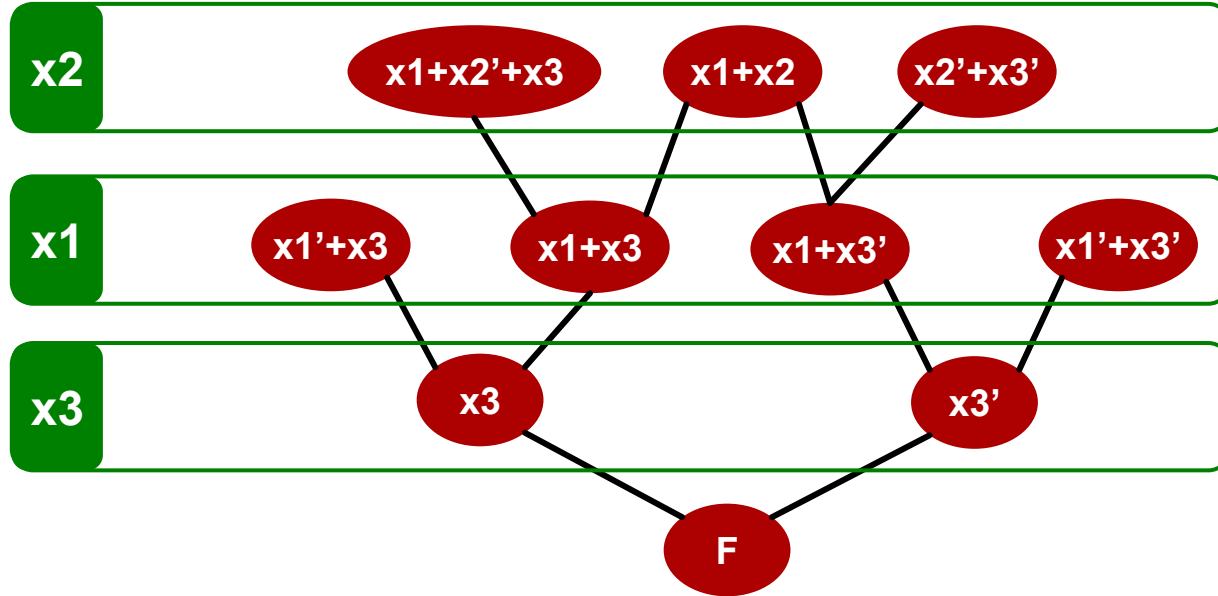
Ordering: $x_2 > x_1 > x_3$



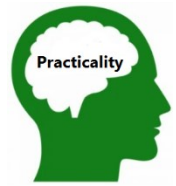
Ordering: $x_2 > x_1 > x_3$



Ordering: $x_2 > x_1 > x_3$



BigSAT: Turning SAT (DP) into Big Data Analytics



- Big Data thinking:

Big Data Solution

||

(1) Large Dataset + (2) Simple Computation + **System Design**

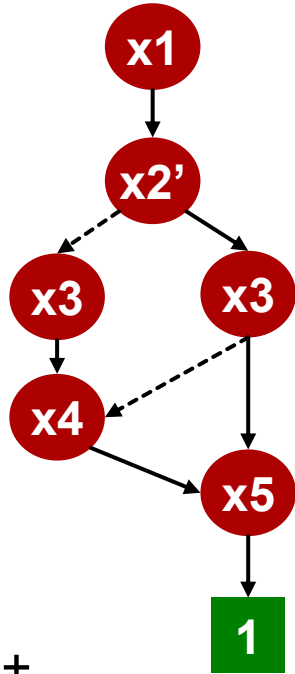
- DP exhibits data parallelism

(1) **Large Num. of Clauses** + (2) **Simple Resolution** + **BigSAT**

ZBDD-based resolution

- ZBDD clauses representation
 - Common prefix and suffix compression
- Multi-resolution on ZBDD
 - Resolution on a pair of sets of clauses
- Clause subsumption elimination

Ordering: $x_1 > x_2 > x_3 > x_4 > x_5$

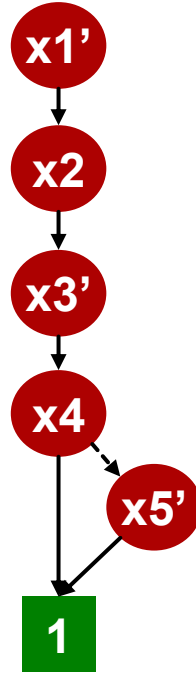


P+

$$(x_1 + x_2' + x_3 + x_5)$$

$$(x_1 + x_2' + x_4 + x_5)$$

$$(x_1 + x_3 + x_4 + x_5)$$

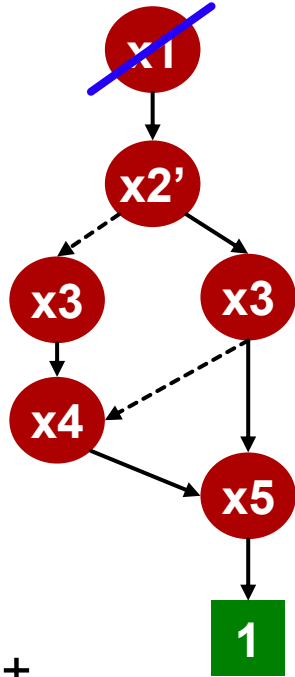


P-

$$(x_1' + x_2 + x_3' + x_4)$$

$$(x_1' + x_2 + x_3' + x_5')$$

Ordering: $x_1 > x_2 > x_3 > x_4 > x_5$



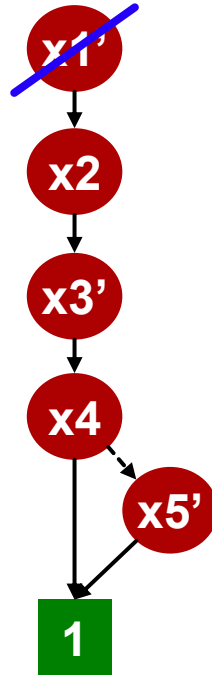
P+

$(x_1 + x_2' + x_3 + x_5)$

$(x_1 + x_2' + x_4 + x_5)$

$(x_1 + x_3 + x_4 + x_5)$

×



P-

$(x_1' + x_2 + x_3' + x_4)$

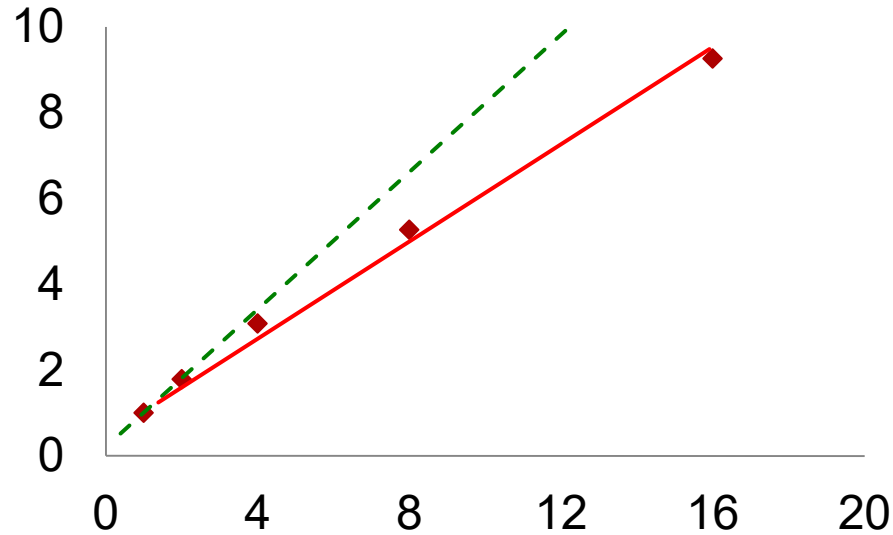
$(x_1' + x_2 + x_3' + x_5')$



()

BigSAT-parallel

- Good scalability factor



- Incremental DP

BigSAT-distributed

- Bulk Synchronous Parallel DP
 - Do resolutions as soon as possible
 - Do resolutions on all buckets
- Load balancing
 - Skewed join on Spark



Conclusion

- **“Big data” thinking to solve problems that do not appear to generate big data**
- **Two example problems**
 - **Interprocedural static analysis**
 - **SAT solving**
- **Future problems**
 - **Symbolic execution**
 - **Program synthesis**
 - **...**