Improved Generic Algorithms for 3-Collisions

Antoine Joux and Stefan Lucks

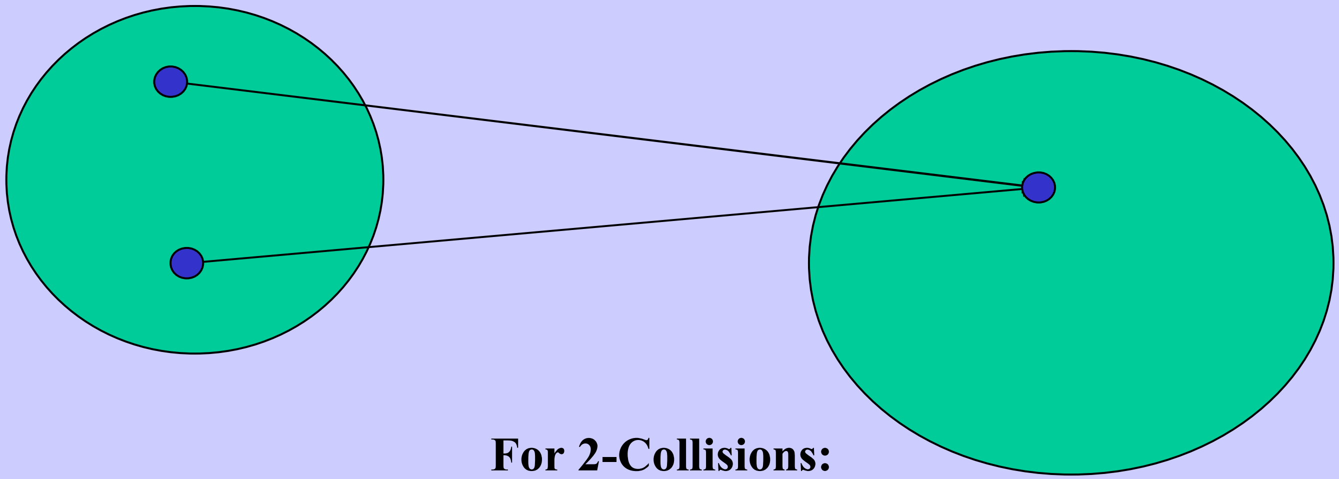
Alon Dayan, 06.06.2012

Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- New Algorithm: 3-Collisions
- Better Tradeoff Algorithm: 3-Collisions
- Parallel Algorithm: 3-Collisions
- General Parallel Algorithm: r-Collisions
- Practical Example

Reminder: 2-Collisions

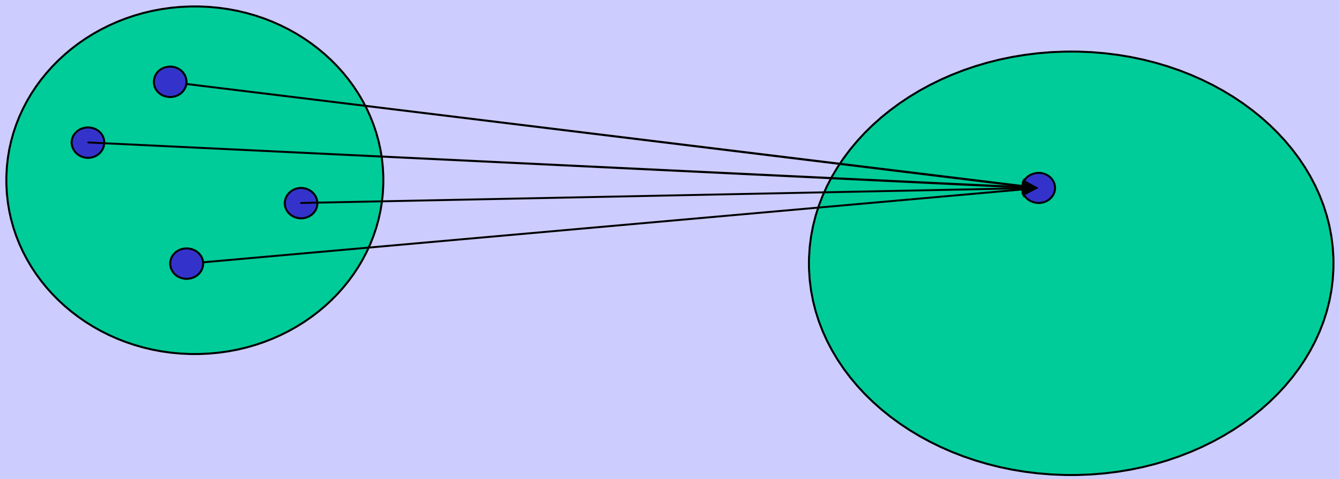
Collision: distinct Inputs with identical outputs (Random Mapping)



For 2-Collisions:

- **Solutions: Floyd/Brent/Nivasch cycle finding algorithms**
- **Negligible memory**
- **Parallelize using distinguished point methods**

What about r-Collisions?



- **Unsolved so far**
- **Usually assume memory:** $N^{\frac{r-1}{r}}$
- **Neglect Parallelization**

Are there memory-efficient and parallelizable algorithms for r-collisions?

Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- **Known Algorithm: 3-Collisions**
- New Algorithm: 3-Collisions
- Better Tradeoff Algorithm: 3-Collisions
- Parallel Algorithm: 3-Collisions
- General Parallel Algorithm: r-Collisions
- Practical Example

Known Algorithm: 3-Collision

Two Main Stages

Stage one:

Initialize with images

for i :1 to N {

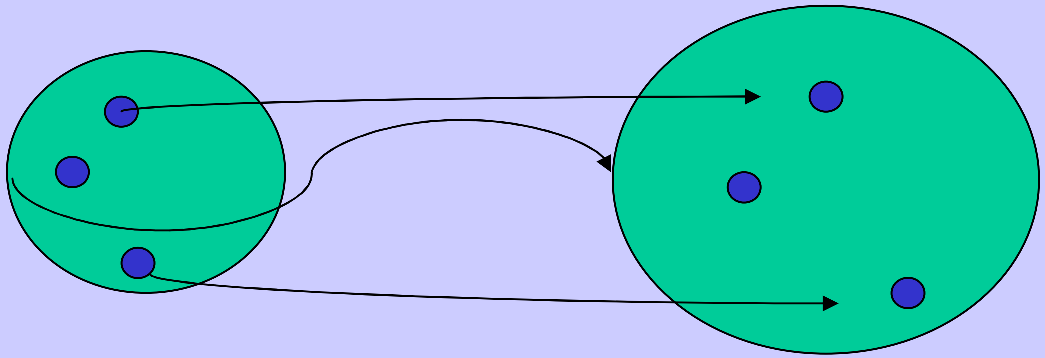
a = Rand [0;N - 1]

b = F(a)

Pr1[i] = a

Img[i] = b

}



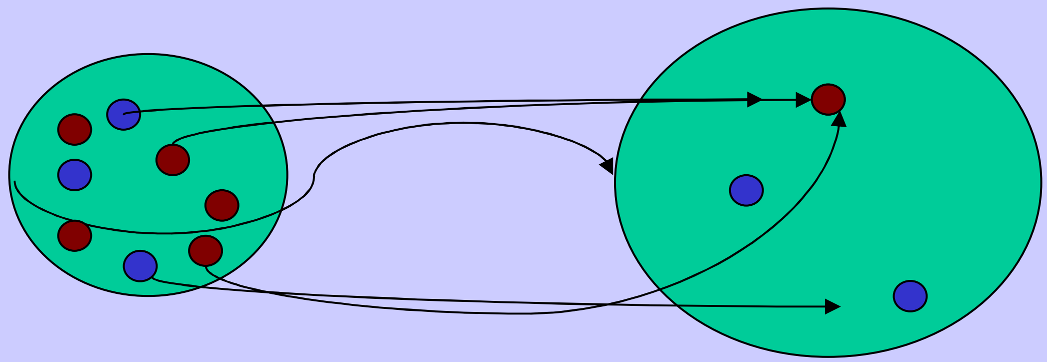
Pr1:	3	6	…	
Pr2:	?	?	…	
Img:	e	t	…	

Known Algorithm: 3-Collision

Two Main Stages

Stage two:

Find collisions



Pr1:	3	6	…	
Pr2:	?	8	…	
Img:	e	t	…	

```
for i: 1 to N {
      c  = Rand [0;N - 1]
      d =  F(a)
      if (d in Img)
          if Pr1[j] != a
              if Pr2[j] == ?
                  then Pr2[j]  = a
          else if Pr2[j] != a
                  3-Collision (Pr1[j]; Pr2[j]; a)
                  Exit
}
```

Known Algorithm: 3-Collision

Two Main Stages

Stage one:

Initialize with images

```
for i :1 to N {
    a  = Rand [0;N - 1]
   b =  F(a)
   Pr1[i]  = a
   Img[i]  =  b
}
```

1

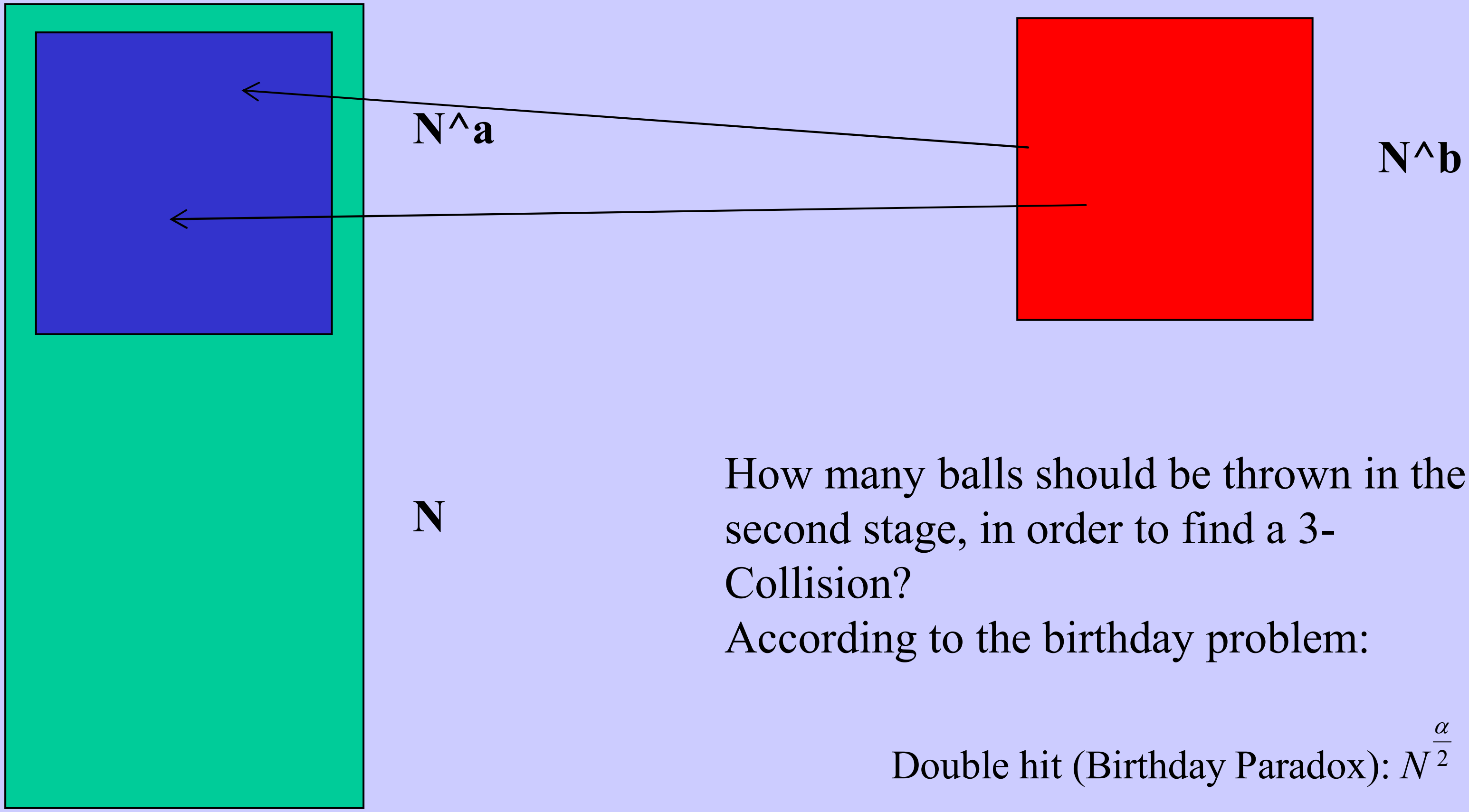
+2

```
for i: 1 to N {
        c  = Rand [0;N - 1]
        d =  F(a)
        if (d in Img)
            if Pr1[j] != a
                if Pr2[j] == ?
                  then Pr2[j]  = a
           else if Pr2[j] != a
                3-Collision (Pr1[j]; Pr2[j]; a)
                Exit
}
```

Stage two:

Find collisions

Known Algorithm: 3-Collision



How many balls should be thrown in the second stage, in order to find a 3-Collision?
According to the birthday problem:

Double hit (Birthday Paradox): $N^{\frac{\alpha}{2}}$

Known Algorithm: 3-Collision

$$\text{Memory}: \tilde{O}\left(N^{\alpha}\right)$$
$$\text{Time}: \tilde{O}\left(N^{\beta}\right)$$
$$\alpha \le \beta$$

$$\tilde{O}\left(g(n)\right) \equiv O\left(g(n) \cdot \log\left(g(n)\right)^{k}\right)$$

$$\text{First stage}: N^{\alpha}$$
$$\text{Second stage}: N^{\beta}$$
$$\text{Expected hits: } N^{\alpha+\beta-1}$$

$$\text{Double hit (Birthday Paradox): } N^{\frac{\alpha}{2}}$$

$$N^{\alpha+\beta-1} \ge N^{\frac{\alpha}{2}}$$
$$\alpha+\beta-1 \ge \frac{\alpha}{2}$$
$$2\alpha+2\beta-2 \ge \alpha$$
$$\alpha+2\beta \ge 2$$
$$\alpha+2\beta = 2$$

if $\alpha=\beta$, then $\alpha=\beta=\frac{2}{3}$. So time and memory: $\tilde{O}\left(N^{\frac{2}{3}}\right)$.

if $\alpha=\frac{1}{2}$, then $\beta=\frac{3}{4}$. So time: $\tilde{O}\left(N^{\frac{3}{4}}\right)$. memory: $\tilde{O}\left(N^{\frac{1}{2}}\right)$.

if constant memory: $\alpha=0$, then $\beta=1$. So time: $\tilde{O}(N)$. memory: $\tilde{O}(1)$.

Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- **New Algorithm: 3-Collisions**
- Better Tradeoff Algorithm: 3-Collisions
- Parallel Algorithm: 3-Collisions
- General Parallel Algorithm: r-Collisions
- Practical Example

New Algorithm: 3-Collisions

Two Main Stages

Stage one:

~~**Initialize with images**~~

Initialize with collisions

```
for i :1 to N {
    a  = Rand [0;N - 1]
    b =  F(a)
    Pr1[i]  = a
    Img[i]  =  b
}
```

HOW?

1

2

```
for i: 1 to N {
        c  = Rand [0;N - 1]
        d =  F(a)
        if (d in Img)
            if Pr1[j] != a
                if Pr2[j] == ?
                    then Pr2[j]  = a
            else if Pr2[j] != a
                    3-Collision (Pr1[j]; Pr2[j]; a)
                    Exit
}
```

SAME

Stage two:

Find third collision

New Algorithm: 3-Collisions

Two Main Stages

Stage one:

~~**Initialize with images**~~

Initialize with collisions

Use cycle finding algorithm

2

1

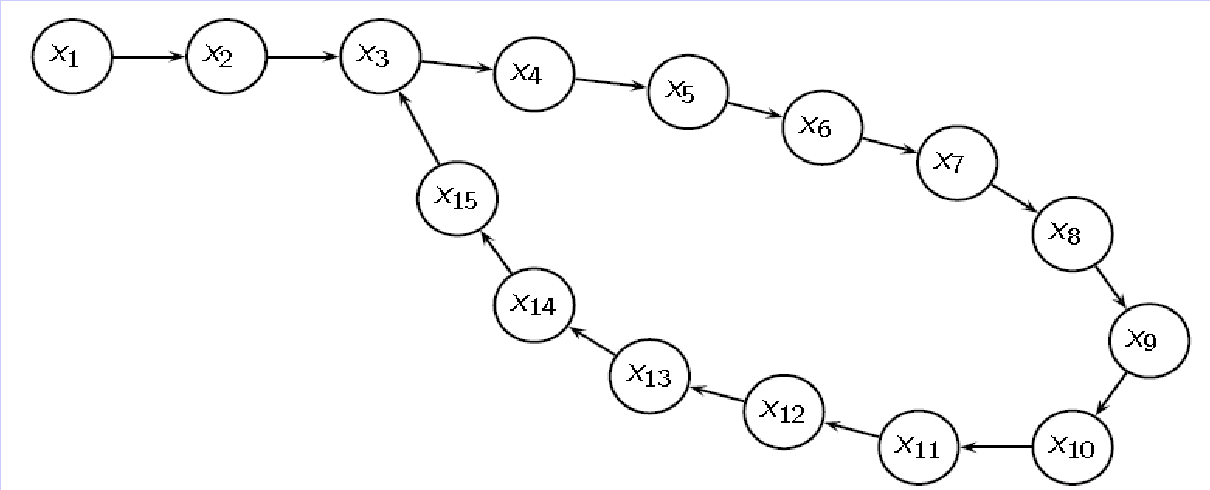
```
for i: 1 to N {
        c  = Rand [0;N - 1]
        d =  F(a)
        if (d in Img)
            if Pr1[j] != a
                if Pr2[j] == ?
                    then Pr2[j]  = a
            else if Pr2[j] != a
                    3-Collision (Pr1[j]; Pr2[j]; a)
                    Exit
}
```

SAME

Stage two:

Find third collision

New Algorithm: 3-Collisions



Floyd's Cycle Finding Algorithm:
p1 is incremented each time by 1 position and p2 is incremented each time by 2 positions until they collide.

Nivasch's Algorithm for Cycle Finding:
reduce the time significantly, while using a small amount of memory.
uses only one pointer.

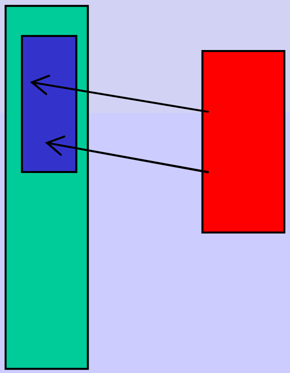
New Algorithm: 3-Collisions

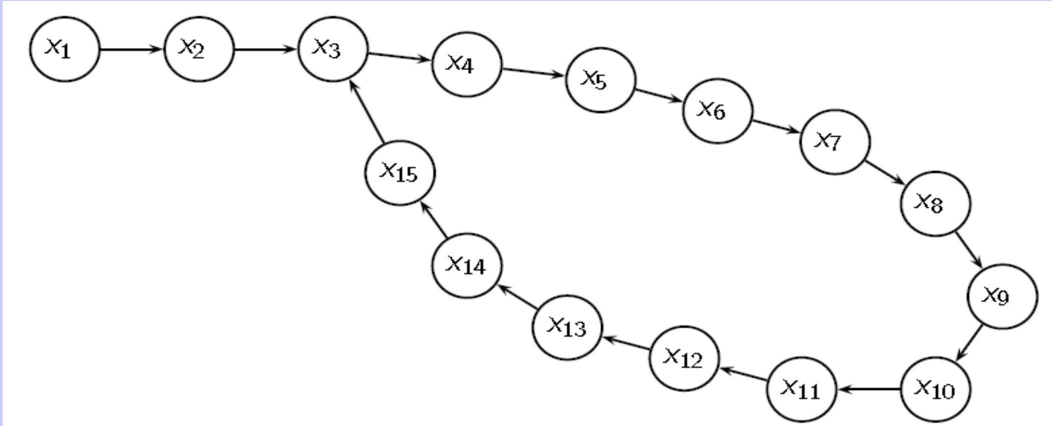
Number of initialized collisions: N^{α}

Time to find each collision: $N^{\frac{1}{2}}$

Time of first step (initialization): $N^{\alpha} \cdot N^{\frac{1}{2}} = N^{\alpha+\frac{1}{2}}$

Second stage : N^{β} (Time)





How many balls should be thrown in the second stage, in order to find a 3-Collision?

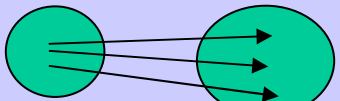
$$\alpha + \beta = 1$$

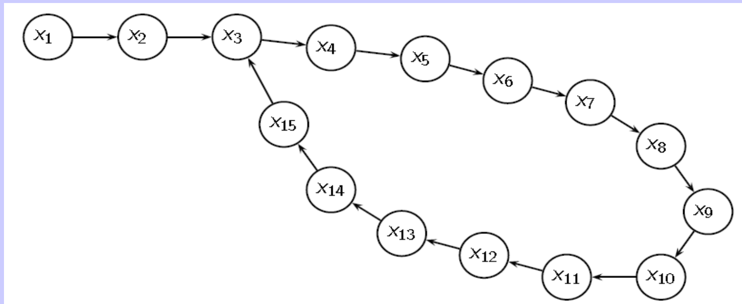
Comparison
3-Collisions Algorithms

Known Algorithm | **New Algorithm**

Tradeoff: $\alpha + 2\beta = 2$ | $\alpha + \beta = 1$

First Stage:





Complexity:

Are there memory-efficient and parallelizable algorithms for r-collisions?

if $\alpha = \beta$, then $\alpha = \beta = \frac{2}{3}$. Time and memory: $\tilde{O}\left(N^{\frac{2}{3}}\right)$.

if $\alpha = \frac{1}{2}$, then $\beta = \frac{3}{4}$. Time: $\tilde{O}\left(N^{\frac{3}{4}}\right)$. Memory: $\tilde{O}\left(N^{\frac{1}{2}}\right)$.

if constant memory: $\alpha = 0$, then $\beta = 1$. Time: $\tilde{O}(N)$. Memory: $\tilde{O}(1)$.

LIMIT

if $\alpha = \frac{1}{4}$, then $\beta = \frac{3}{4}$. Time: $\tilde{O}\left(N^{\frac{3}{4}}\right)$. Memory: $\tilde{O}\left(N^{\frac{1}{4}}\right)$.

Improved Generic Algorithms for 3-Collisions

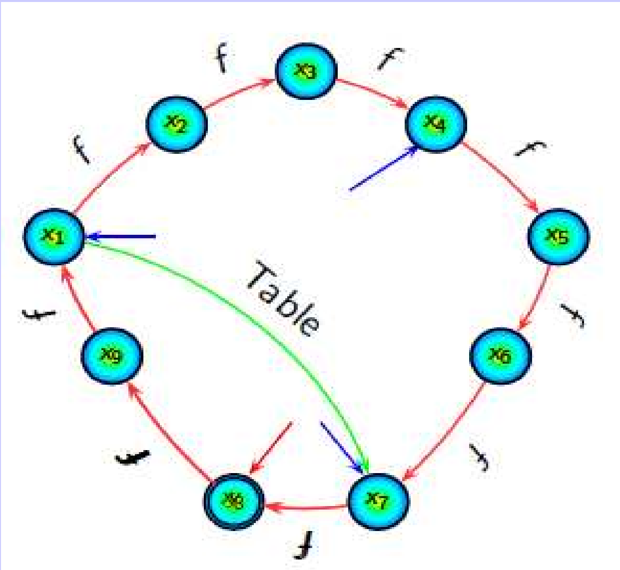
- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- New Algorithm: 3-Collisions
- **Better Tradeoff Algorithm: 3-Collisions**
- Parallel Algorithm: 3-Collisions
- General Parallel Algorithm: r-Collisions
- Practical Example

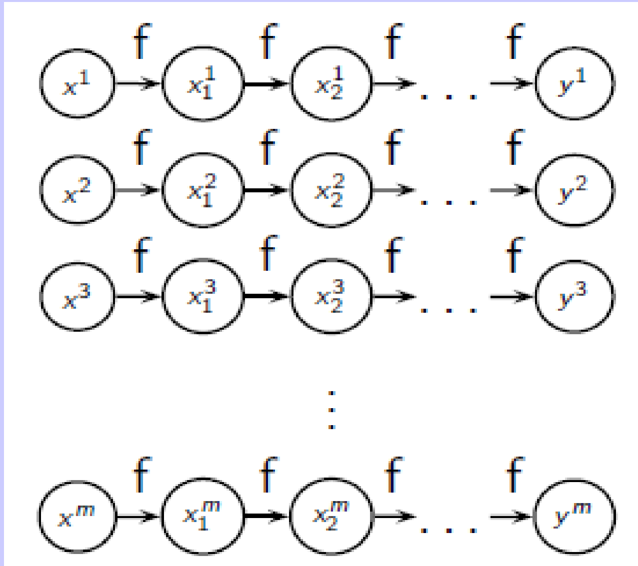
Better Tradeoff Algorithm: 3-Collisions

Change First Stage

Goal: $N^{\frac{1}{3}}$ collisions in time: $\tilde{O}\left(N^{\frac{2}{3}}\right)$, and memory: $\tilde{O}\left(N^{\frac{1}{3}}\right)$

Use Hellman's time-memory tradeoff:



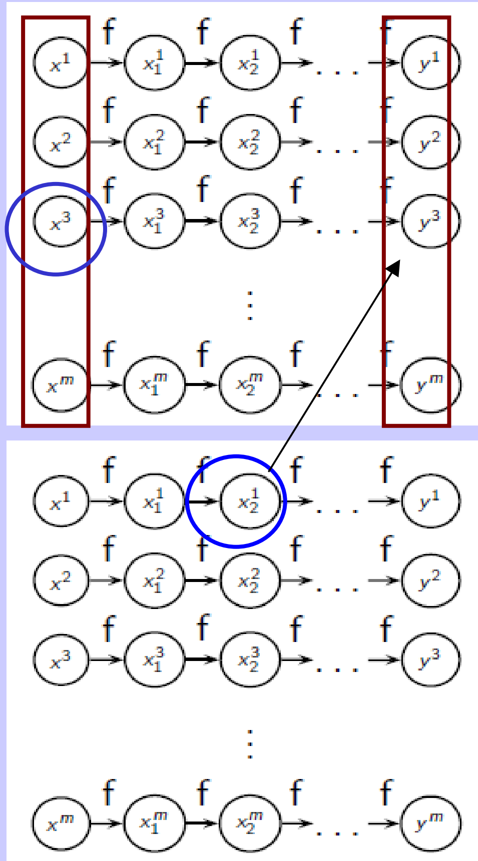


Better Tradeoff Algorithm: 3-Collisions

Hellman's time-memory tradeoff:

Build N^{α} chains of length N^{γ} (Start: Random Position, N^{γ} iterations of F)

Save start and end points
Sort (by end points)
  Build N^{α} new chains of length N^{γ}
But check for merge after each F
Backtracking for collision



Possible with only one set of chains
Check each F with previous end points

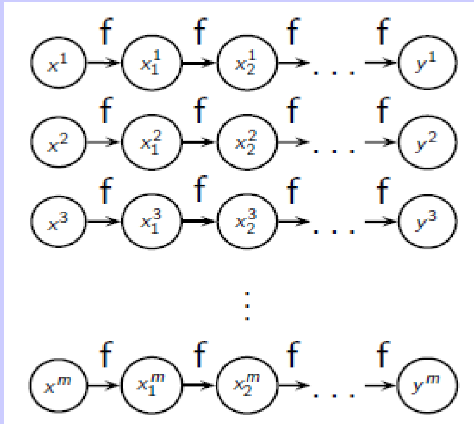
Cost: Harder to implement. Same order

Better Tradeoff Algorithm: 3-Collisions

Expected number of collisions: $\mathrm{O}\left(N^{2\alpha+2\gamma-1}\right)$

Aim: N^{α} collisions

$$N^{\alpha}\cdot N^{\gamma} = N^{\alpha+\gamma}$$



Set: $\gamma=\dfrac{(1-\alpha)}{2}$

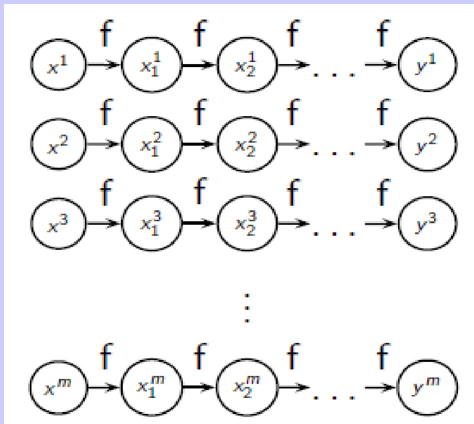
Running Time: $N^{\alpha+\gamma} = N^{\alpha+\frac{(1-\alpha)}{2}} = N^{\frac{(1+\alpha)}{2}}$

Since $\alpha \le \dfrac{1}{3}$: $\dfrac{(1+\alpha)}{2} \le 1-\alpha$

Time: $\tilde{\mathrm{O}}\left(N^{\frac{2}{3}}\right)$: dominated by second step

Memory: $\tilde{\mathrm{O}}\left(N^{\frac{1}{3}}\right)$

$$N^{\alpha}\cdot N^{\gamma} = N^{\alpha+\gamma}$$



Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- New Algorithm: 3-Collisions
- Better Tradeoff Algorithm: 3-Collisions
- **Parallel Algorithm: 3-Collisions**
- General Parallel Algorithm: r-Collisions
- Practical Example

Parallel Algorithm: 3-Collisions

Are there memory-efficient and parallelizable algorithms for r-collisions?

Algorithms until now – badly suited to parallelization

Problem: Replication. Large amount of memory on every processor

$$\text{Aim}: \ \sim N^{\frac{1}{3}} \text{ processors, time: } \tilde{O}\left(N^{\frac{1}{3}}\right), \text{ constant memory (each processor)}$$

Efficient communication between processors for

Small transmitted data

Parallel Algorithm: 3-Collisions

Distinguished Points

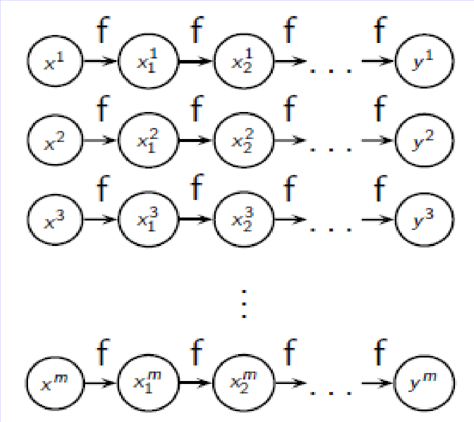
Set of points + efficient procedure for deciding membership

N numbers

Set of distinguished points: [0, M-1] $0 \le x \le M - 1$

Fraction of distinguished points: $\dfrac{M}{N}$

Parallel Algorithm: 3-Collisions



In our case:

Average Chain Length: $N^{\frac{1}{3}}$

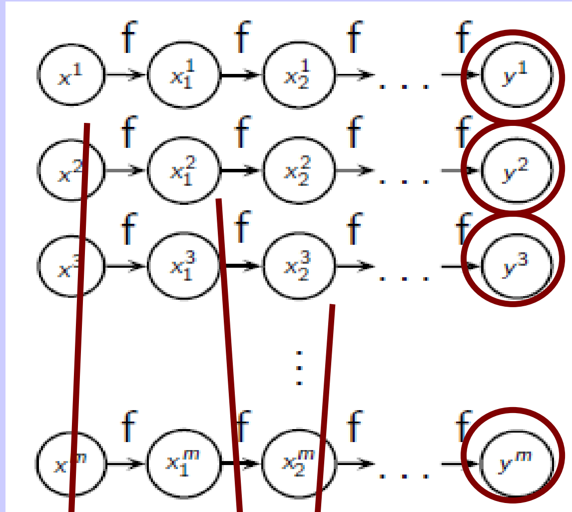
Choose $M =\sim N^{\frac{2}{3}}$ distinguished points

Parallel Algorithm: 3-Collisions

Step 1:

- Each processor starts from a random position (s)
- Iteratively apply F
- Until d (distinguished point) is encountered
- length from s to d is L
- Send (s, d, L) to processor #d (mod number of processors)

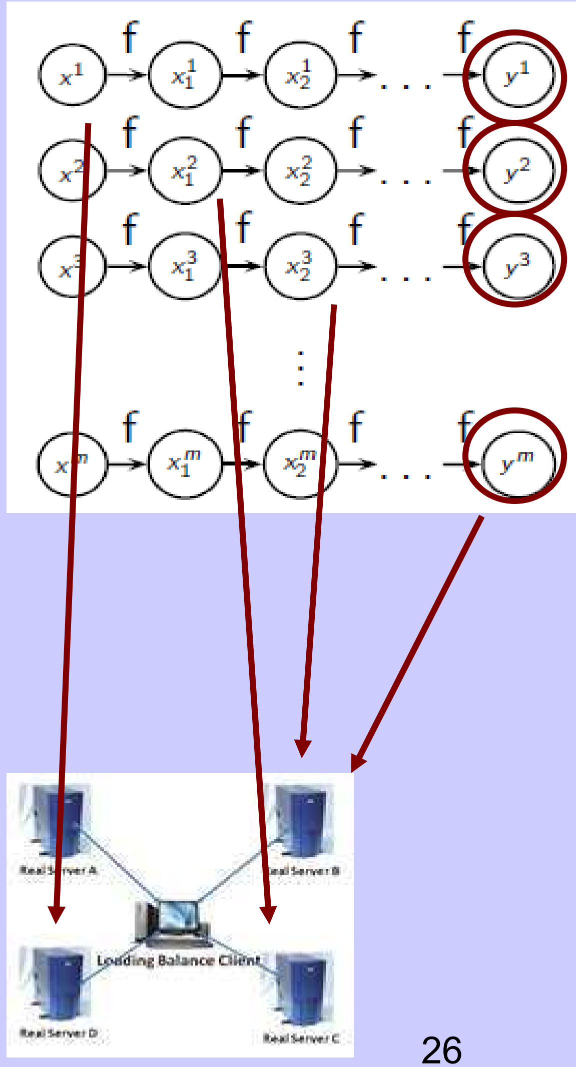
(Abort after a reasonable amount of time if did not find)



Parallel Algorithm: 3-Collisions

Step 2:

- Each processor received triplets (s, d, L)
- If a specific d appears three times (or more):
- Synchronize the chains (using L(length))
- If merged at a common position –
  3-collision found



Parallel Algorithm: 3-Collisions

Partial Parallelization is also possible to find 3-Collisions

#Processors(Q) < $N^{\frac{1}{3}}$

Time: O*(N^(2/3-Q))

Each processor: local memory O(N^(1/3-Q))

Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- New Algorithm: 3-Collisions
- Better Tradeoff Algorithm: 3-Collisions
- Parallel Algorithm: 3-Collisions
- **General Parallel Algorithm: r-Collisions**
- Practical Example

Extension: r-collision (r>3)

Generalization of previous algorithm (r=3)

For r-collision, number of F evaluations needed: $(r!)^{\frac{1}{r}} \cdot N^{\frac{r-1}{r}}$

Constant for fixed r

Problem: Long chains create too many collisions (between one chain and all others)

Waste time on recomputing same evaluations

Solution: More chains, but shorter $\quad Length \leq N^{\frac{1}{r}}$

Extension: r-collision (r>3)

In General, to find r-collisions, it is best to have:

#Processors: N^((r-2)/r)

#Distinguished points: N^((r-1)/r)

Chains length: N^(1/r)

r triplets sent to same processor: r-Collision found

Partial Parallelization is also possible

For big r's it is important to use parallelization

Improved Generic Algorithms for 3-Collisions

- Reminder: 2-Collisions
- Known Algorithm: 3-Collisions
- New Algorithm: 3-Collisions
- Better Tradeoff Algorithm: 3-Collisions
- Parallel Algorithm: 3-Collisions
- General Parallel Algorithm: r-Collisions
- **Practical Example**

Practical Example

Random function F(x):

Xoring two copies of DES (different keys)

$$F\left(x_{64 \text{ bits}}\right) = DES_{K1}\left(x\right) \oplus DES_{K2}\left(x\right)$$

Known Algorithm- Time: 2^{43}. Memory: 2^{46} bytes = 64 Terabytes.

Another Tradeoff - Time: 2^{48}. Memory: 2^{32} bytes.

Requires a high-end computer (no parallelization)

Parallel - M=2^{44} Distinguished Points, 2^{20} Chains

Found: 35M chains, 3M groups (3 or more chains with same endpoints)

Biggest Group: 36 Chains

Conclusion: Slightly shorter chains needed

94 CPU-days $\rightarrow$ 11.5 CPU-days

Improved Generic Algorithms for 3-Collisions

Thank You

Questions?