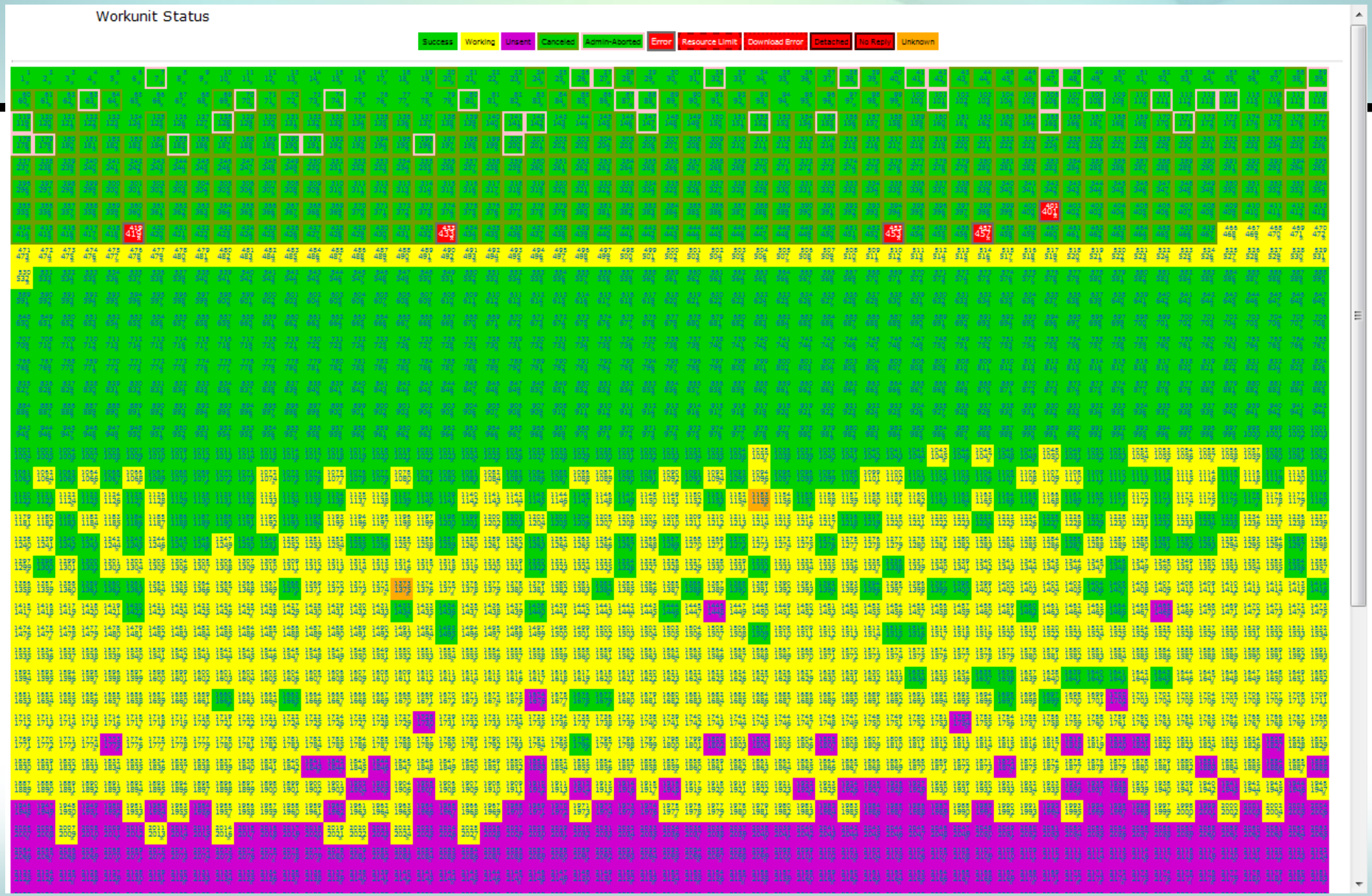
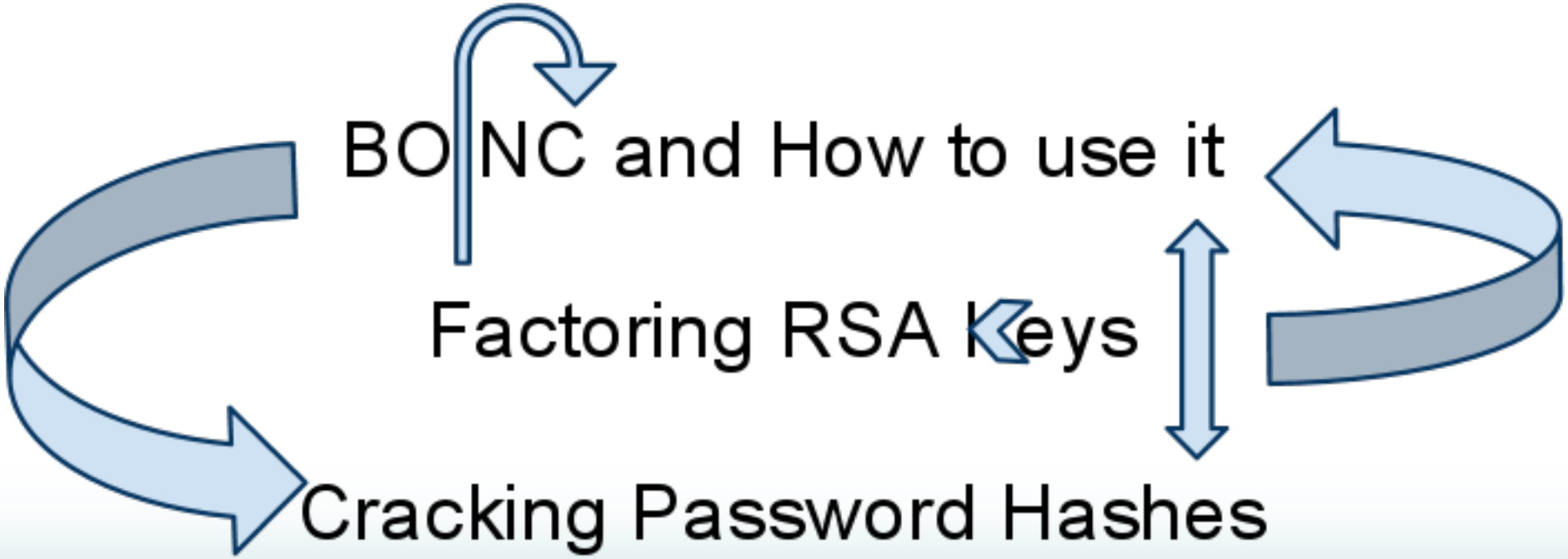
GOTHAM
DIGITAL • SCIENCE

Cloud & Control

Factoring RSA Keys & Cracking Passwords


Workunit Status
Success
Working
Unsent
Canceled
Admin-Aborted
Error
Resource Limit
Download Error
Detached
No Reply
Unknown

- BOINC and How to use it
- Factoring RSA Keys
- Cracking Password Hashes

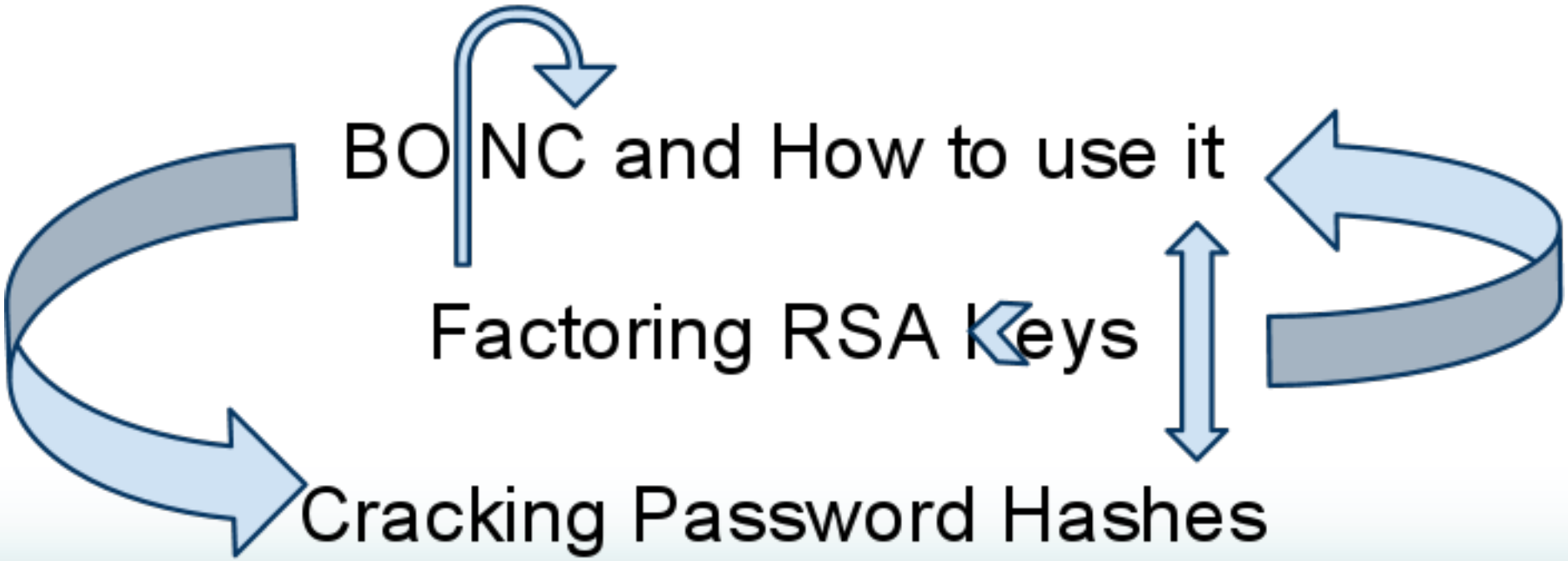


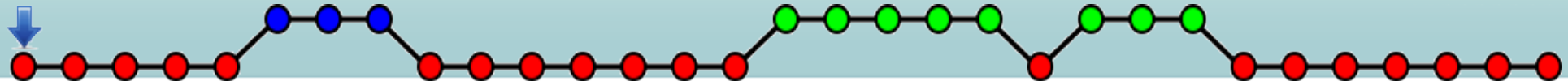
You have interesting problems!

- Fuzzing
- ROP Compilers
- SMT Solving

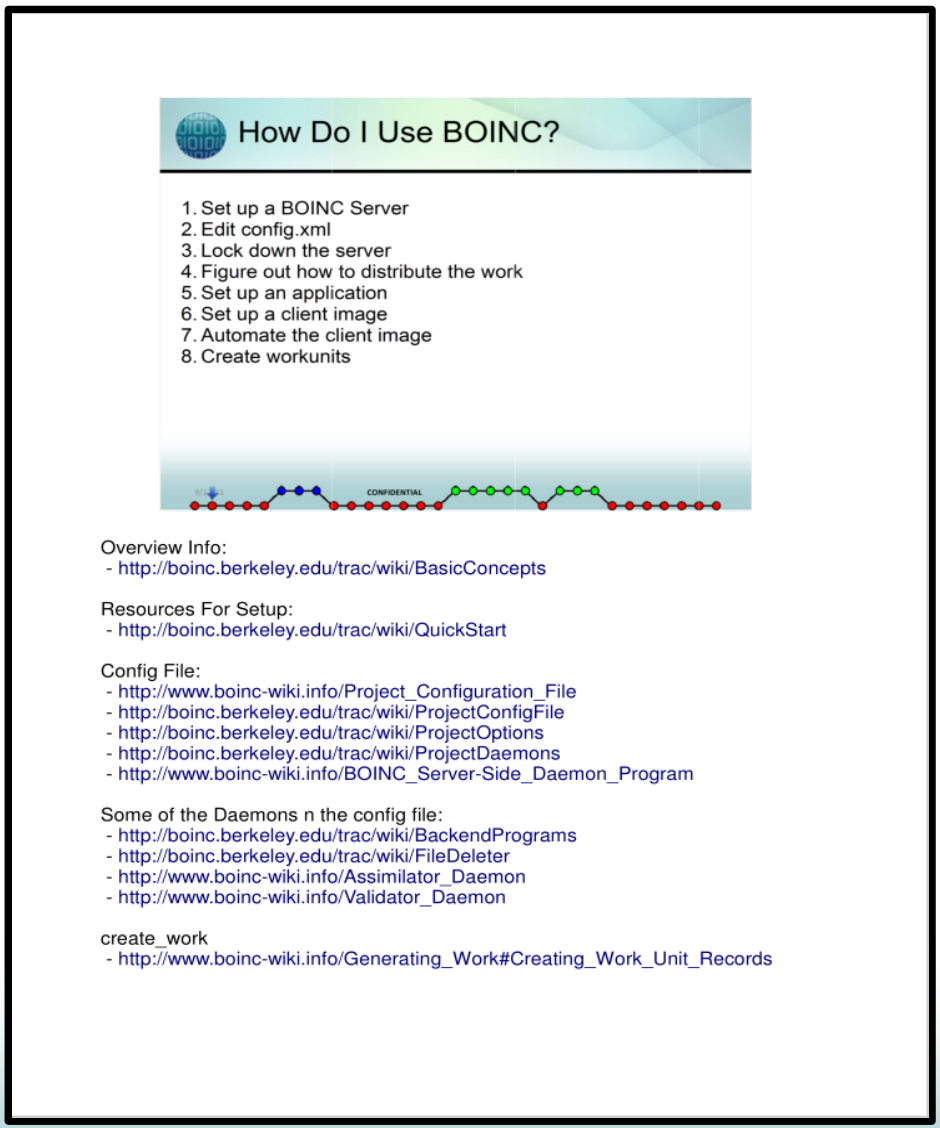
Would BOINC Help?

How would you fit your problem into BOINC?

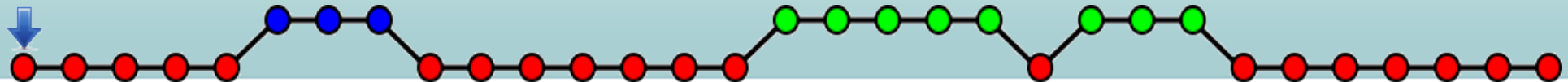




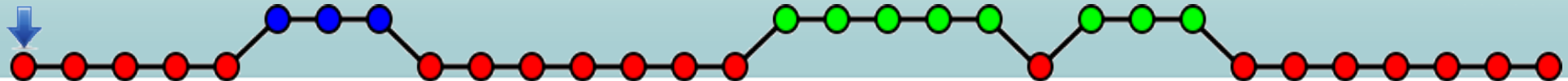
Materials!



- Slides w/ references
- Sample Templates
- Scripts




BOINC



History of BOINC

1999 - SETI@home launches to the public

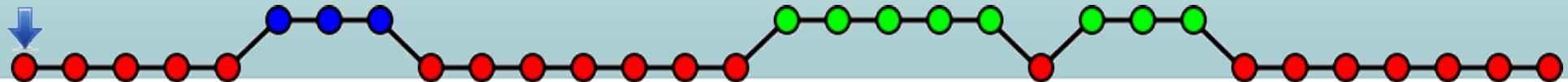
2004 - BOINC project begins
- First BOINC Project launches (protein prediction)

2008 - GPU powered applications introduced

~2 million users
~6 million computers

Top projects (by credit):

1 SETI@home
2 MilkyWay@home
3 Collatz Conjecture
32ish SHA-1 Collision Search



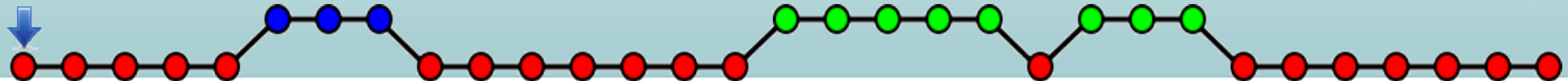
Why would I use it?

Handles

- network problems
- client errors
- server/client reboots
- file integrity

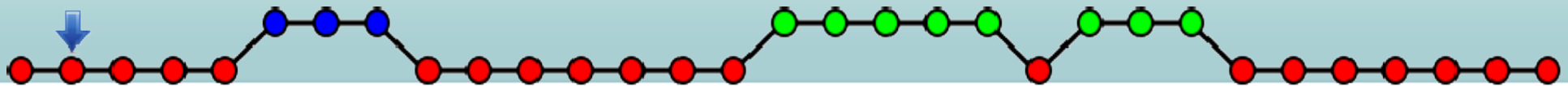
Supports

- running time limits
- multiple platforms
- untrustable clients
- GPUs and odd applications
- credit/reputation & teams
- assimilation/validation



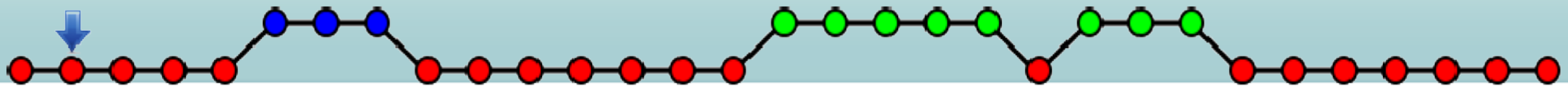
How Do I Use BOINC?

1. Set up a BOINC Server
2. Edit config.xml
3. Lock down the server
4. Figure out how to distribute the work
5. Set up an application
6. Set up a client image
7. Automate the client image
8. Create workunits



Is it hard?

1. Set up a BOINC Server - Easy
2. Edit config.xml - Easy
3. Lock down the server - Should be easy
4. Figure out how to distribute the work - Could be tricky
5. Set up an application - Trial and Error
6. Set up a client image - Easy
7. Automate the client image - Easy
8. Create workunits - Potentially annoying



HACKERS CAN TURN YOUR HOME COMPUTER INTO A BOMB

. . . & blow your family to smithereens!

By RANDY JEFFRIES / *Weekly World News*

WASHINGTON — Right now, computer hackers have the ability to turn your home computer into a bomb and blow you to Kingdom Come — and they can do it anonymously from thousands of miles away!

Experts say the recent "break-ins" that paralyzed the Amazon.com, Buy.com and eBAY websites are tame compared to what will happen in the near future.

Computer expert Arnold Yabenson, president of the Washington-based consumer group National CyberCrime Prevention Foundation (NCPF), says that as far as computer crime is concerned, we've only seen the tip of the iceberg.

"The criminals who knocked out those three major online businesses are the least of our worries," Yabenson told *Weekly World News*.

"There are brilliant but unscrupulous hackers out there who have developed technologies that the average person can't even dream of. Even people who are familiar with how computers work have trouble getting their minds around the terrible things that can be done.

"It is already possible for an assassin to send someone an e-mail with an innocent-looking attachment connected to it. When the receiver downloads the attachment, the electrical current and molecular structure of the central processing unit is altered, causing it to blast apart like a large hand grenade.

"As shocking as this is, it shouldn't surprise anyone. It's just the next step in an ever-escalating progression of horrors conceived and instituted by hackers."

Yabenson points out that these dangerous sociopaths have already:

- Vandalized FBI and U. S. Army websites.
- Broken into Chinese military networks.
- Come within two digits of cracking an 87-digit Russian security code that would have sent deadly missiles hurtling toward five of America's major cities.

"As dangerous as this technology is right now, it's going to get much scarier," Yabenson said.

"Soon it will be sold to terrorists cults and fanatical religious-fringe groups.

"Instead of blowing up a single plane, these groups will be able to patch into the central computer of a large airline and blow up hundreds of planes at once.

"And worse, this e-mail bomb program will eventually find its way into the hands of anyone who wants it.

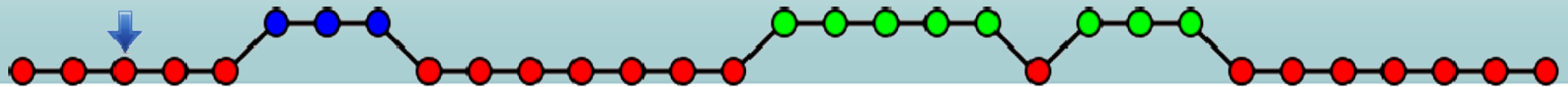
"That means anyone who has a quarrel with you, holds a grudge against you or just plain doesn't like your looks, can kill you and never be found out."

KABOOM! It might not look like it, but an innocent home computer like this one can be turned into a deadly weapon.

Sickos can wreak death and destruction from thousands of miles away!

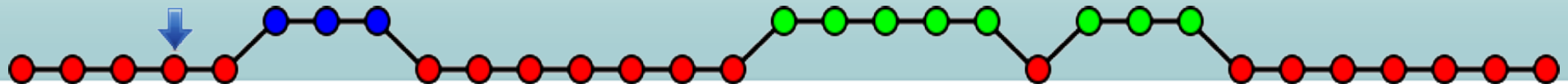
Arnold Yabenson.

who's got two thumbs and isn't responsible for you getting owned?



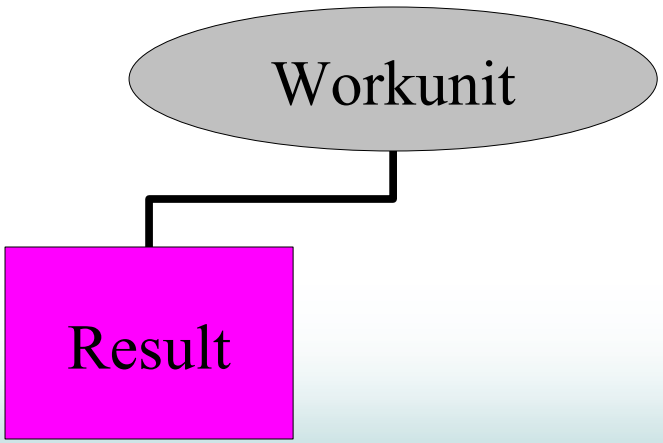
Lifecycle of a unit of work

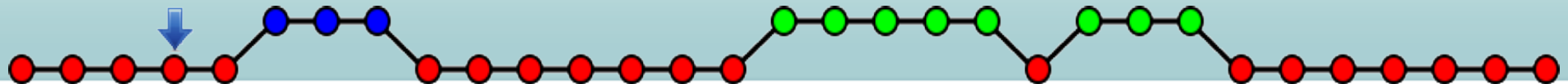
1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion



Lifecycle of a unit of work

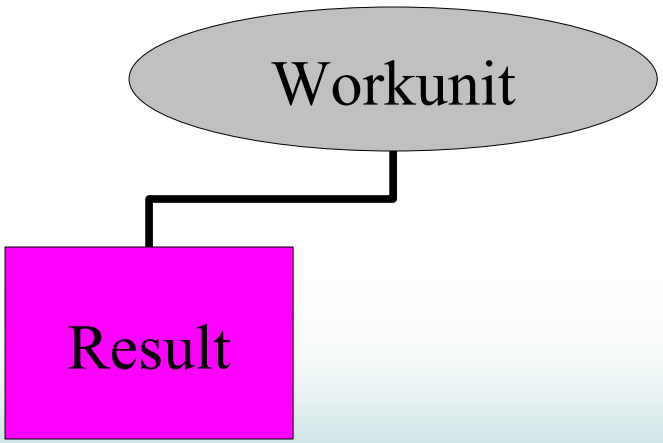
1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion

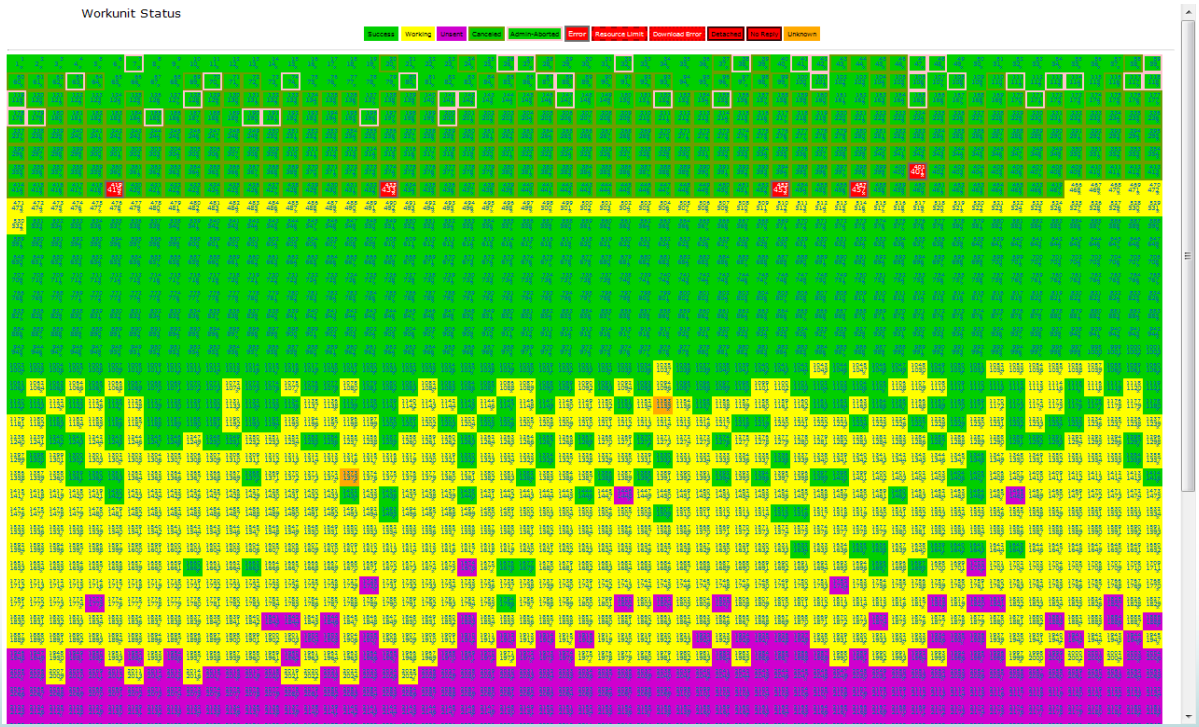


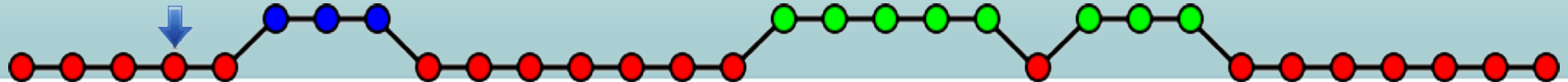


Lifecycle of a unit of work

1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion

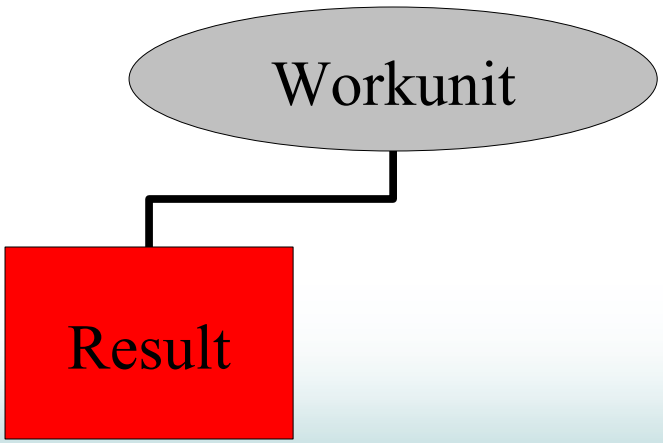


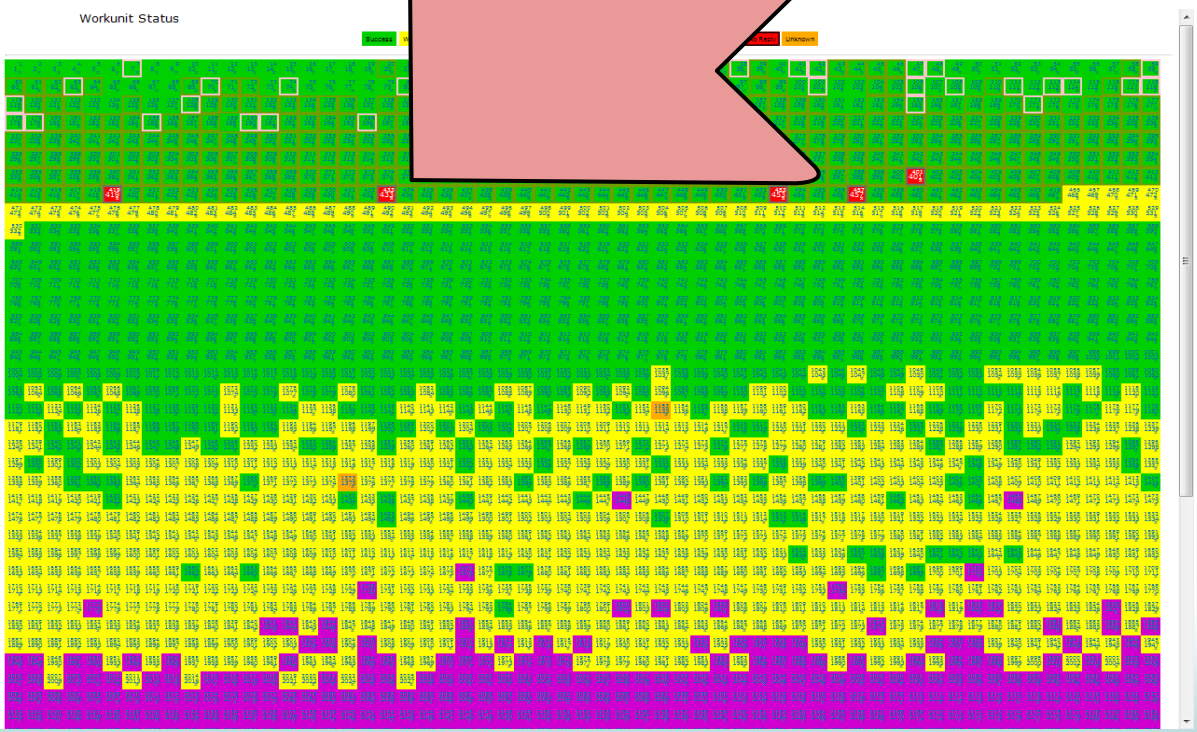


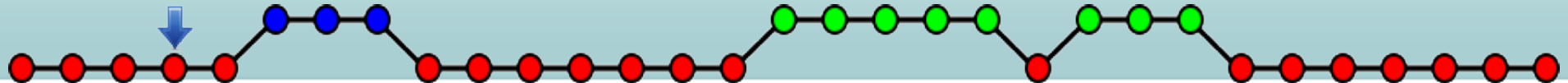


Lifecycle of a unit of work

1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'res[illegible]s'
5. Client computes and upload the outc[illegible]esult'
6. Server Validation
7. Server Assimilation
8. Server Deletion

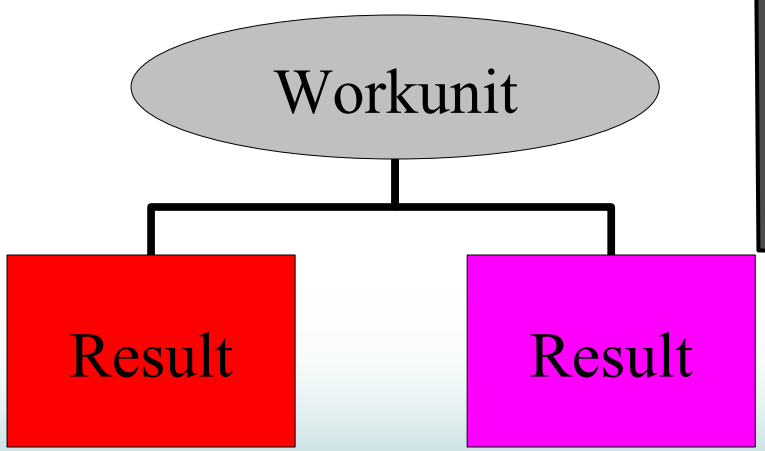


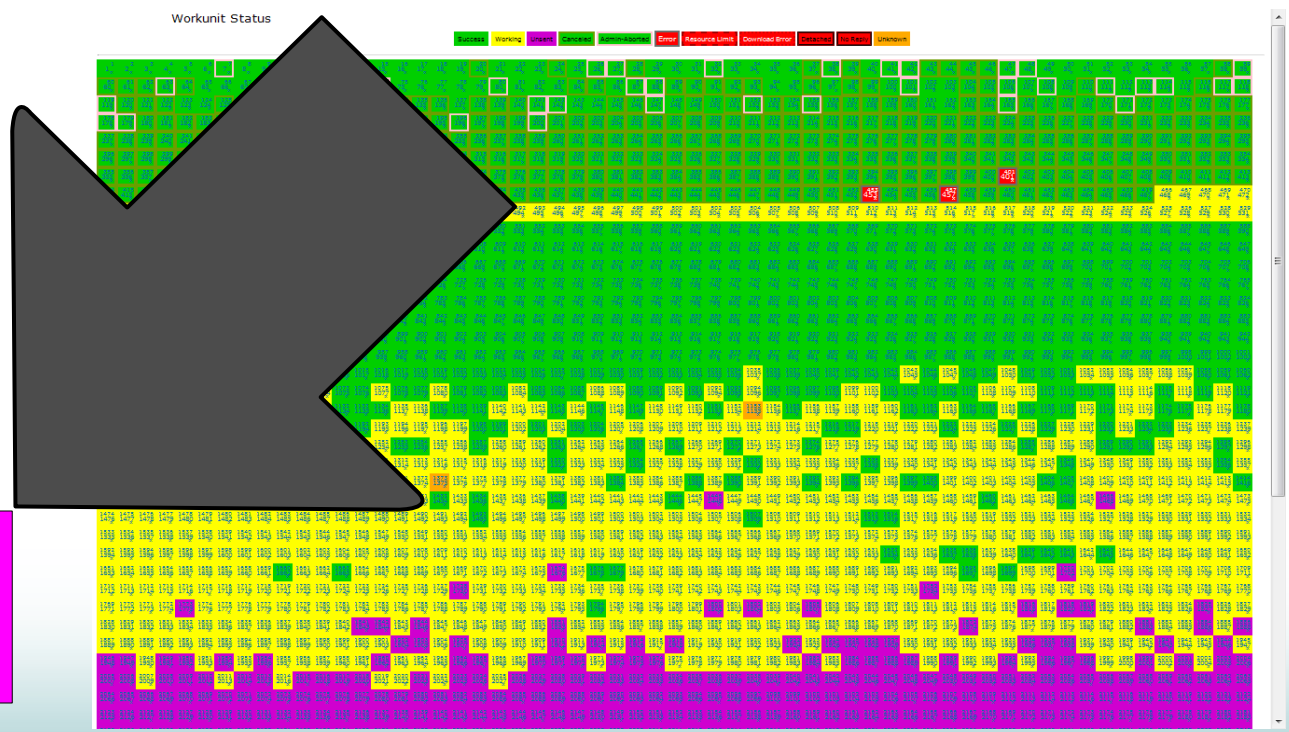


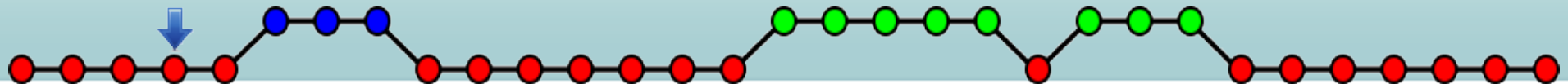


Lifecycle of a unit of work

1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion

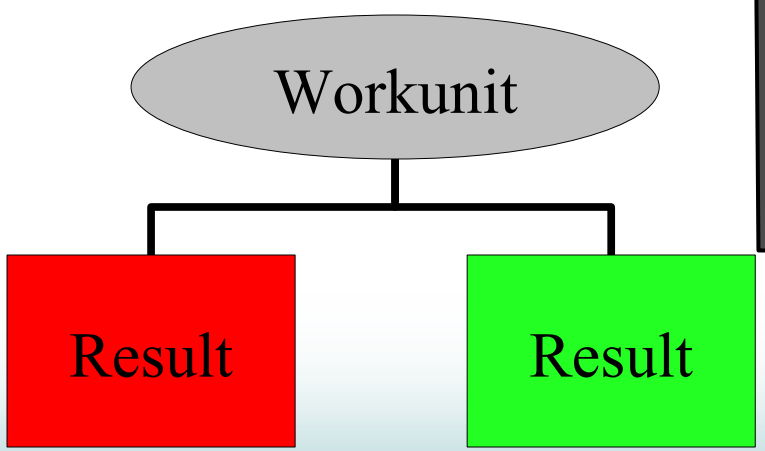


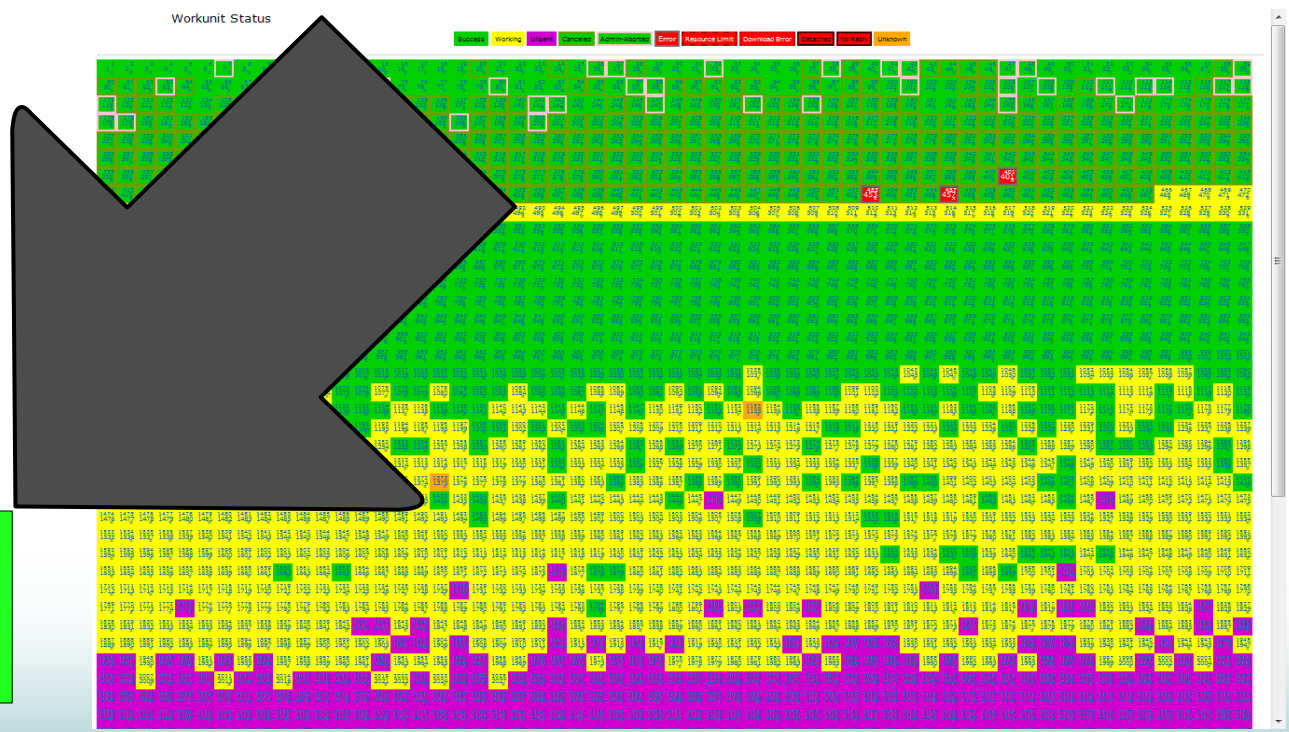


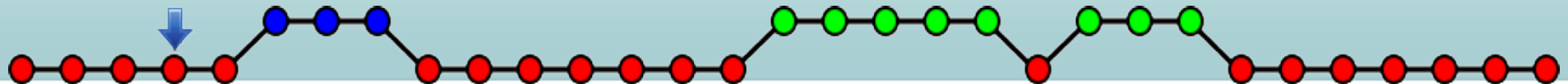


Lifecycle of a unit of work

1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion





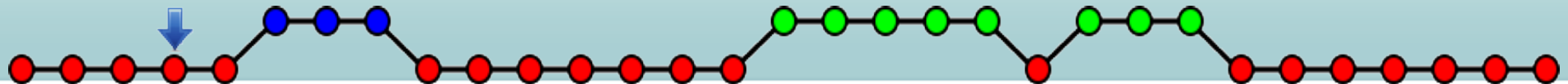


Lifecycle of a unit of work

1. Create boinc workunits
2. Load them into the server
3. Server creates 'results'
4. Client connects and is assigned 'results'
5. Client computes and upload the outcome of the 'result'
6. Server Validation
7. Server Assimilation
8. Server Deletion

Can actually be really complicated!

But for us.... no.
- sample_bitwise_validator
- sample_assimilator

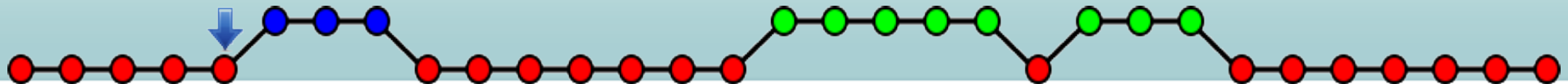


512 Bit RSA Key Factoring

History

p*q = n <- n is a semiprime
5 * 3 = 15 <- 15 is a semiprime
(76-digit p) * (76 digit q) = (155 digit n)

- Aug 1999 - 512 Bit Factored for the first time (publicly)
- 2004 - GGNFS, msieve and factLat.pl in development
- July 2009 - TI83+ Signing Key Factored
- Aug 2009 - Factoring Service Offered: $5000/key
- Sept 2009 - All TI Signing Keys factored
- Dec 2009 - 768 Bit factored for the first time (publicly)
  40 + 1500 + 155 = 1695 Core-Years



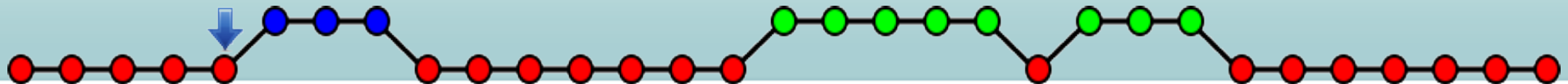
How Do I Factor

1. Trial Division?
   - Is it divisible by 2? 3? 5? 7? 11? 13

2. Pollard Rho

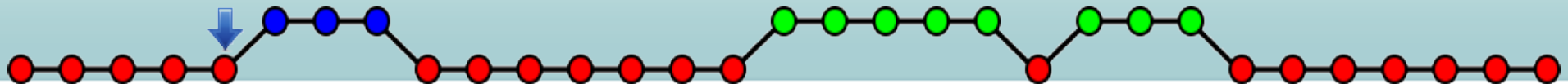
3. ECM

4. General Number Field Sieve



How Do I Factor - GNFS

1. Polynomial Selection
2. Sieving
3. Combine



How Do I Factor - GNFS

1. Polynomial Selection
2. Sieving
3. Combine

1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

How Do I Factor - GNFS

1. Polynomial Selection
2. Sieving
3. Combine

1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

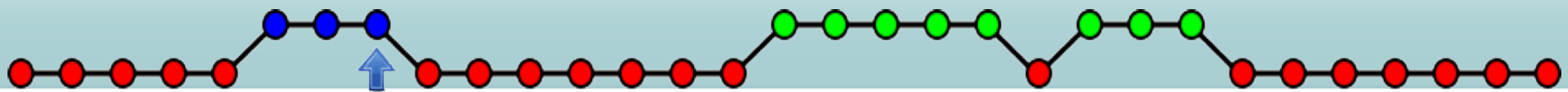
1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving

How Do I Factor - GNFS

1. Polynomial Selection
2. Sieving
3. Combine

1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving



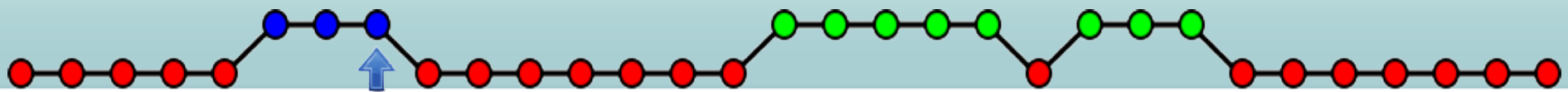
How Do I Factor - GNFS

1. Polynomial Selection
2. Sieving
3. Combine

1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving

1. Filter Relations & Build Matrix
2. Linear Algebra using Lanczos
3. "Square Root Phase"



How Do I Factor – GNFS

1. Polynomial Selection
2. Sieving
3. Combine

Slow & Unparallelizable

512 Bit ~8 Core-Days

768 Bit ~155 Core-Years*

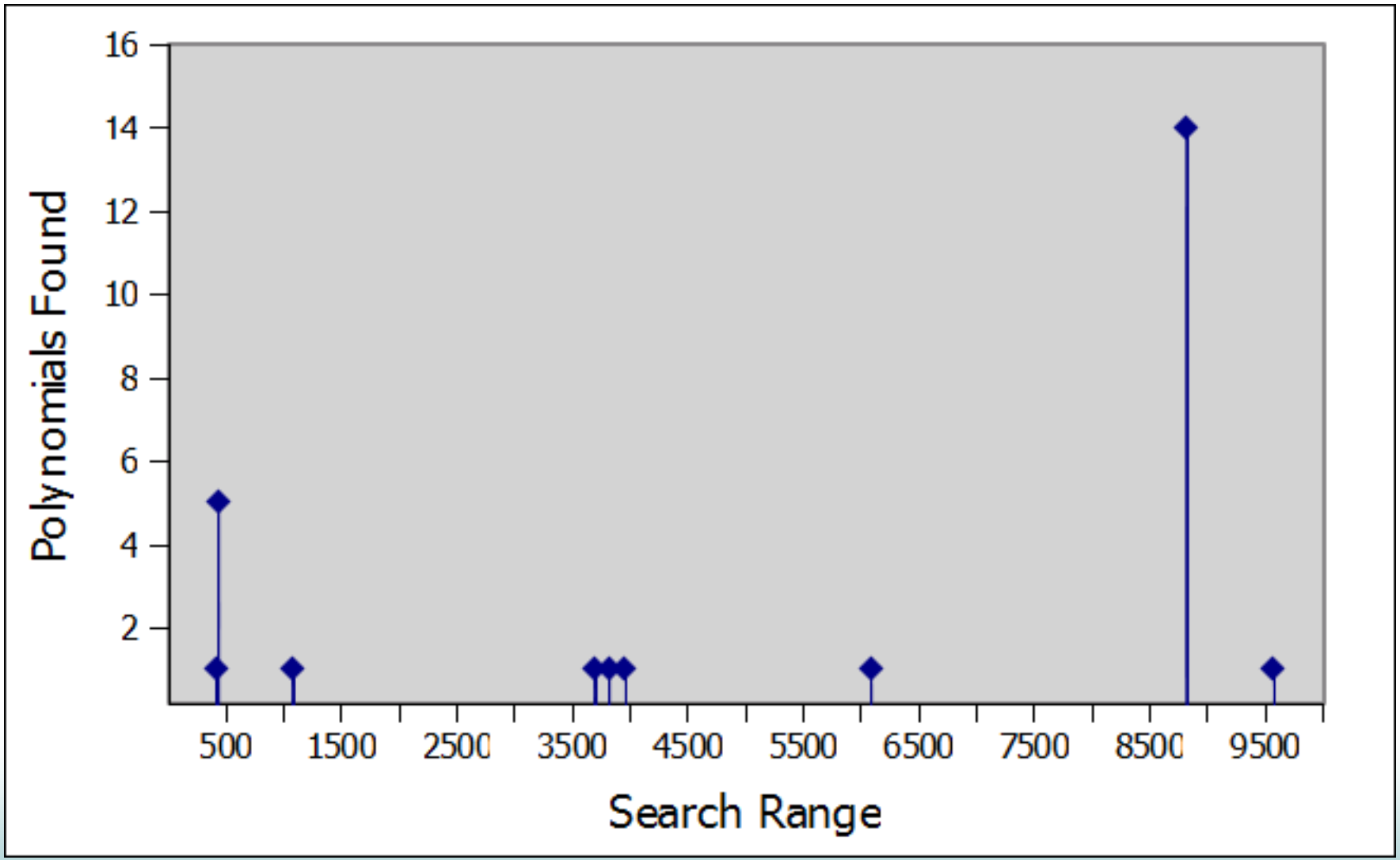
1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving

1. Filter Relations & Build Matrix
2. Linear Algebra using Lanczos
3. "Square Root Phase"

How Do I Factor

1. Polynomial Selection



msieve by jasonp

Beautiful C Code
All Factoring Algorithms
- Trial Division
- Phollard Rho
- ECM
- GNFS

Actively Developed & Maintained
Active Support Channel
Active Community

(& happy ending)

Polynomial Selection

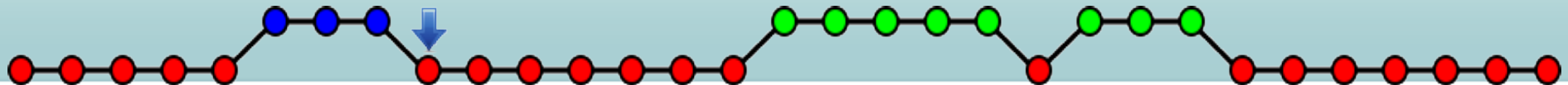
1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

Sieving

1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving

Combine

1. Filter Relations & Build Matrix
2. Linear Algebra using Lanczos
3. "Square Root Phase"



msieve by jasonp

jasonp?

Polynomial Selection

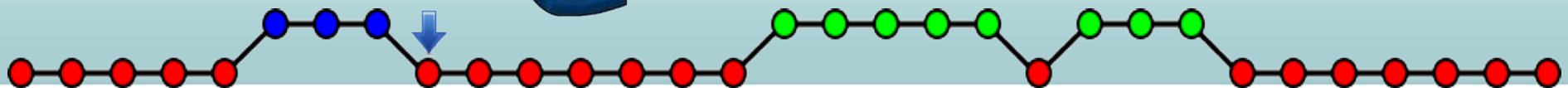
1. f(x) & g(x) of degree d, e
2. irreducible over rationals
3. interpreted mod n have common root mod m

Sieving

1. Millions of pairs a,b
2. Such that $b^d \cdot f(a/b)$ & $b^e \cdot g(a/b)$ factor 'prettily' (are smooth)
3. Via Lattice Sieving

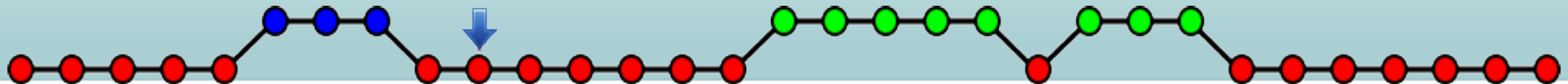
Combine

1. Filter Relations & Build Matrix
2. Linear Algebra using Lanczos
3. "Square Root Phase"



BOINC-izing an Open Source App

- fopen -> boinc_fopen
- boinc_init()
  boinc_finish(return_value)
- link with boinc libs

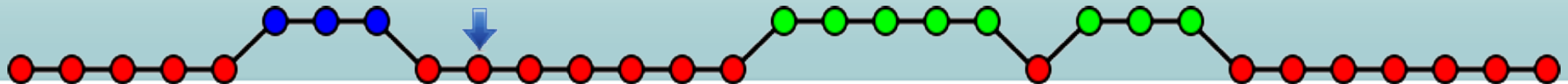


BOINC-izing an Open Source App

- fopen -> boinc_fopen
- boinc_init()
  boinc_finish(return_value)
- link with boinc libs

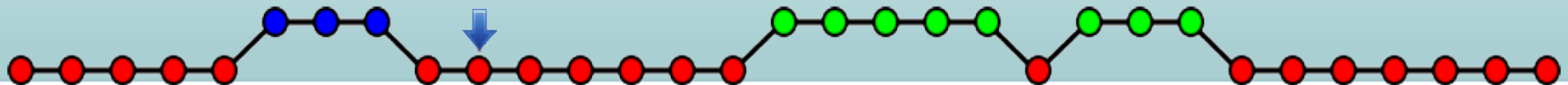
Optional:

- Checkpointing
- Critical Sections
- boinc_fraction_done
- boinc_need_network



Rewiring msieve into a BOINC Application

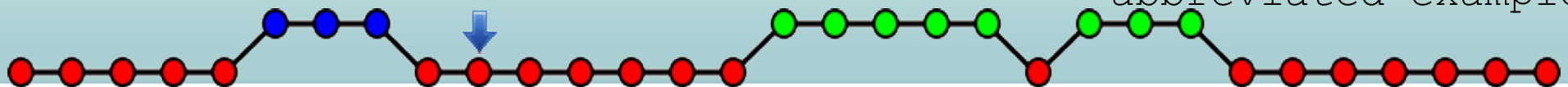
```
@@ -2852,7 +2891,33 @@
+#ifdef HAVE_BOINC
 int main(int argc, char **argv)
+{
+ int newArgc, ret;
+ char** newArgv;
+ myboincstart(&newArgc, &newArgv, argv[0]);
+ ret = sieve_main(newArgc, newArgv);
+ boinc_finish(ret);
+ return ret;
+}
+
+
+int sieve_main(int argc, char **argv)
+#else
+int main(int argc, char **argv)
+#endif
 {
```



Rewiring msieve into a BOINC Application

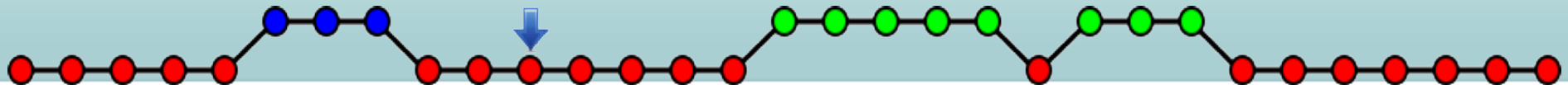
```
void myboincstart(int* argc, char *** argv, char* name)
{
    char in[500], out[500];
    boinc_init();
    boinc_resolve_filename("input_data", in, 500);
    boinc_resolve_filename("output_data", out, 500);

    *argc = 0;
    argv = new char*[7];
    argv[(*argc)++] = name;
    argv[(*argc)++] = "-i";
    argv[(*argc)++] = in;
    argv[(*argc)++] = "-nf";
    argv[(*argc)++] = out;
    argv[(*argc)++] = "-np";
    argv[(*argc)++] = "\0";
}
```

*abbreviated example

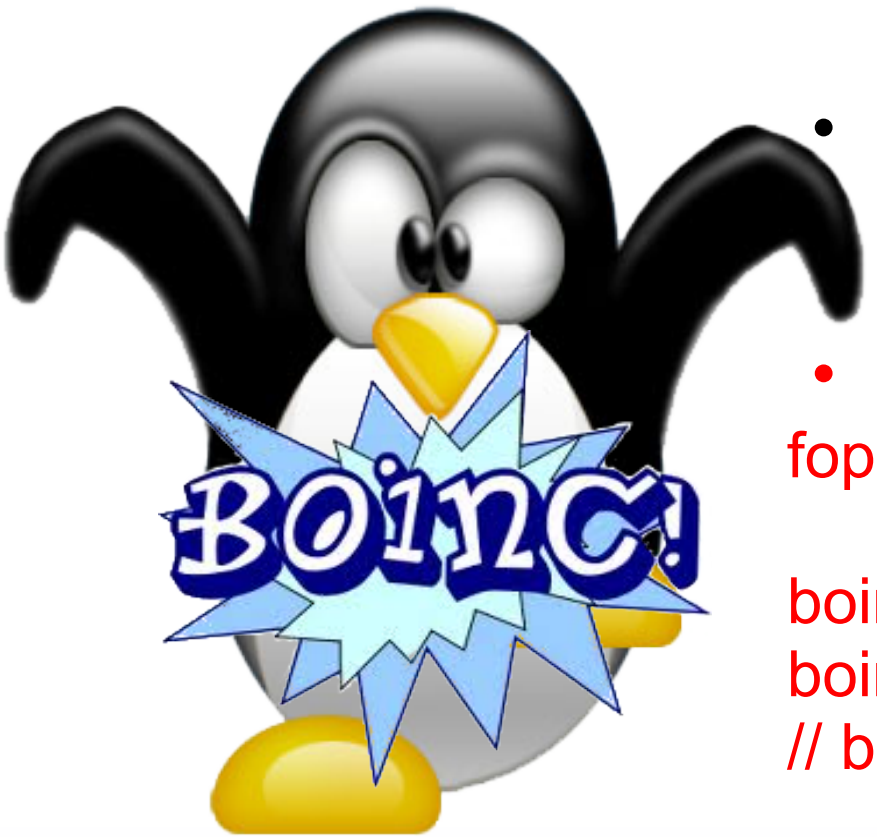


BOINC-izing an Open Source App

- fopen -> boinc_fopen
- boinc_init()
  boinc_finish(return_value)
- link with boinc libs



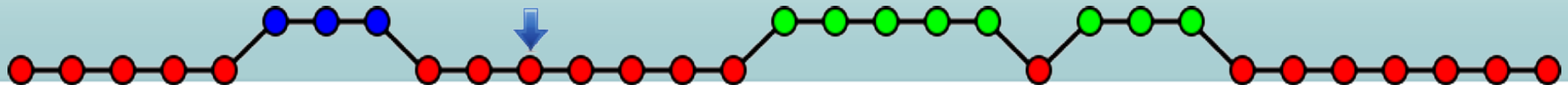
BOINC-izing an Open Source App



- fopen -> boinc_fopen
- boinc_init()
  boinc_finish(return_value)
- boinc_resolve_filename

```
fopen("logfile", "w")

boinc_resolve_filename("logfile", buffer);
boinc_fopen(buffer, "w")
// buffer -> workunit12345_0_1
```

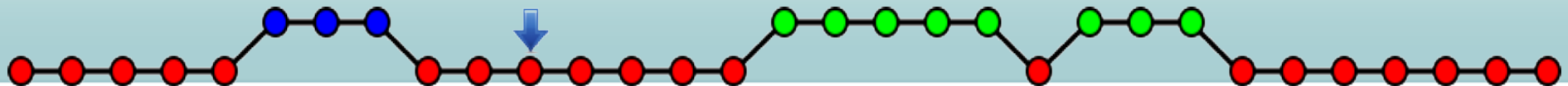
- link with boinc libs



Application Templates

Input

```
<file_info>
    <number>0</number>
     [ <sticky /> ]
     [ <nodelete /> ]
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>rsakey</open_name>
    [ <copy_file/> ]
  </file_ref>

  <target_nresults>1</target_nresults>
</workunit>
```



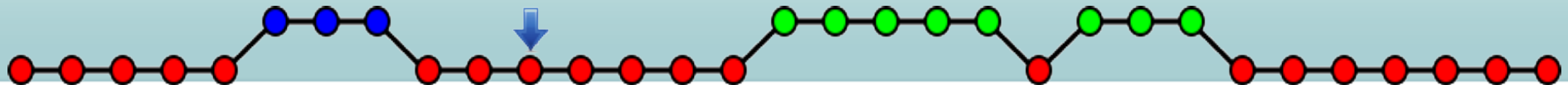
Application Templates

Input

```
<file_info>
    <number>0</number>
     [ <sticky /> ]
     [ <nodelete /> ]
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>rsakey</open_name>
    [ <copy_file/> ]
  </file_ref>

  <target_nresults>1</target_nresults>
</workunit>
```

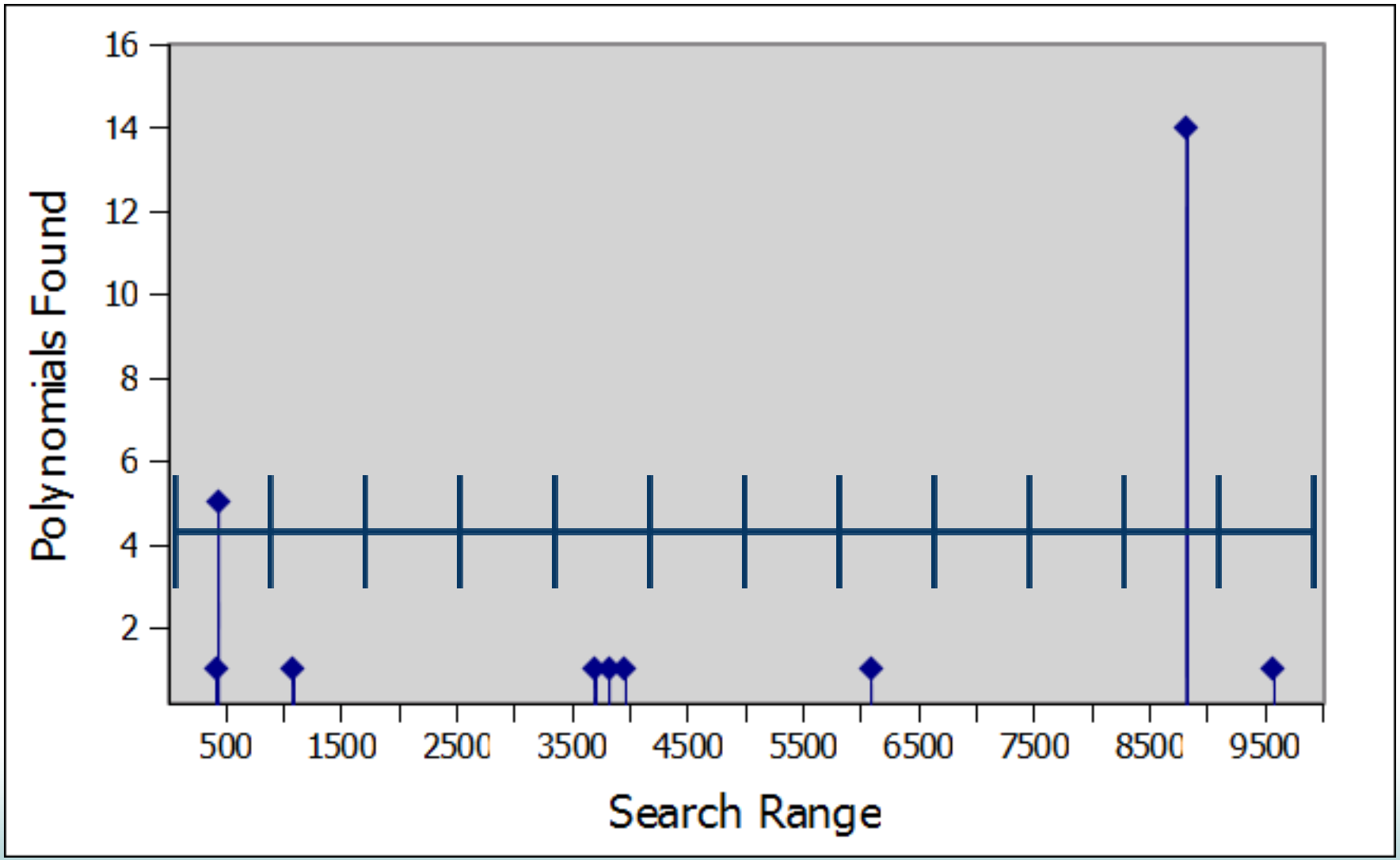
Output

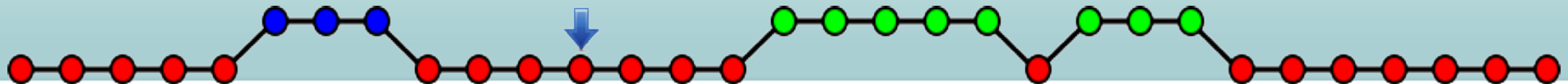
```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
 <file_ref>
  <file_name><OUTFILE_0/>
                </file_name>
 <open_name>logfile</open_name>
 [ <copy_file>0|1</copy_file> ]
 [ <optional>0|1</optional> ]
 </file_ref>
</result>
```



How Do I Factor

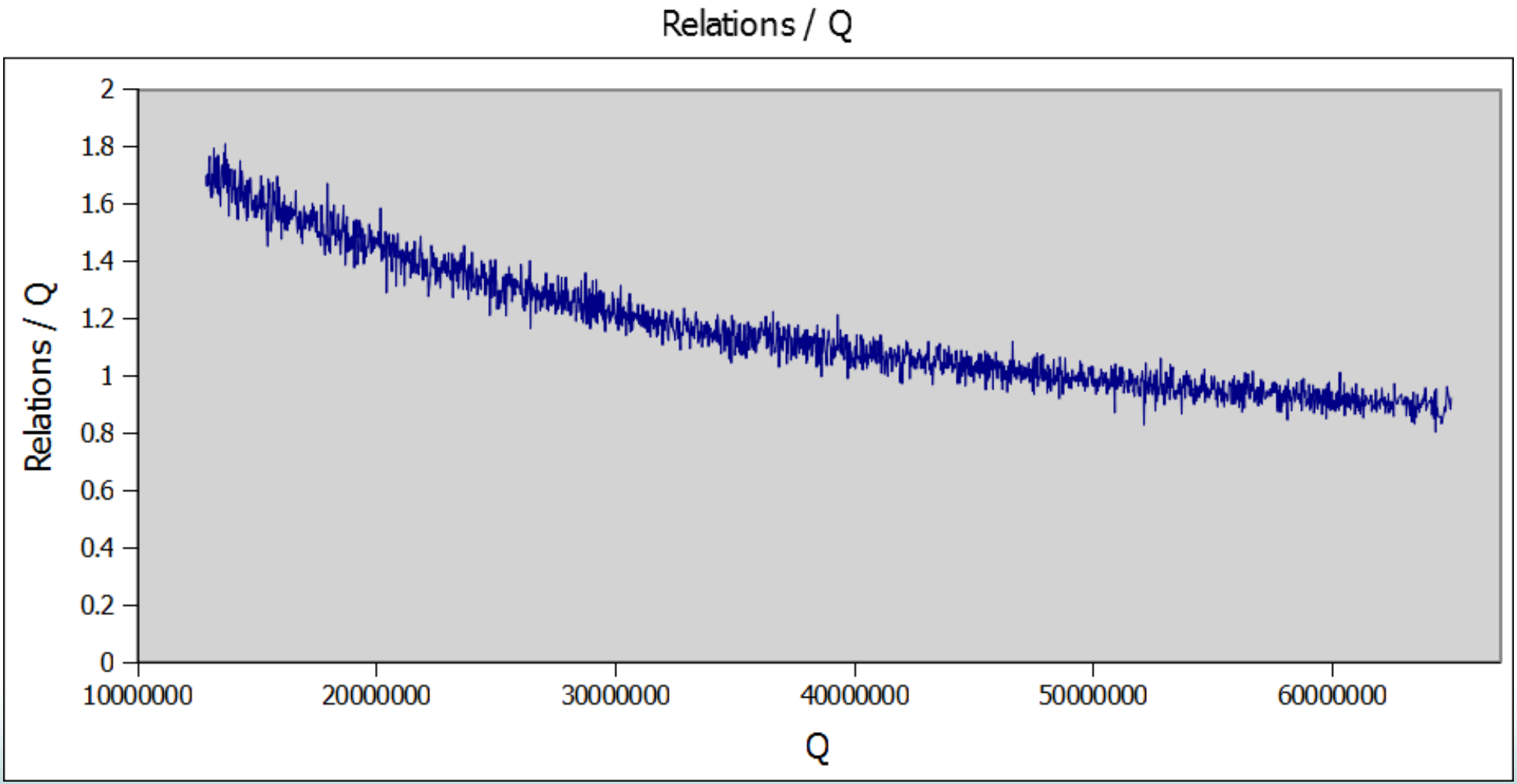
1. Polynomial Selection

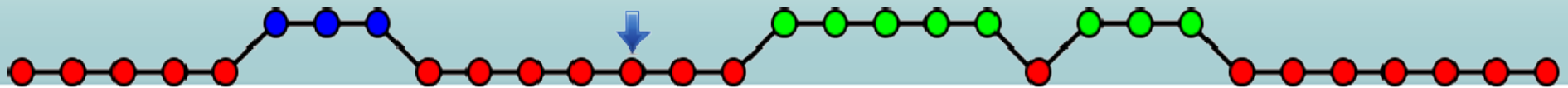




How Do I Factor

1. Polynomial Selection
2. Sieving



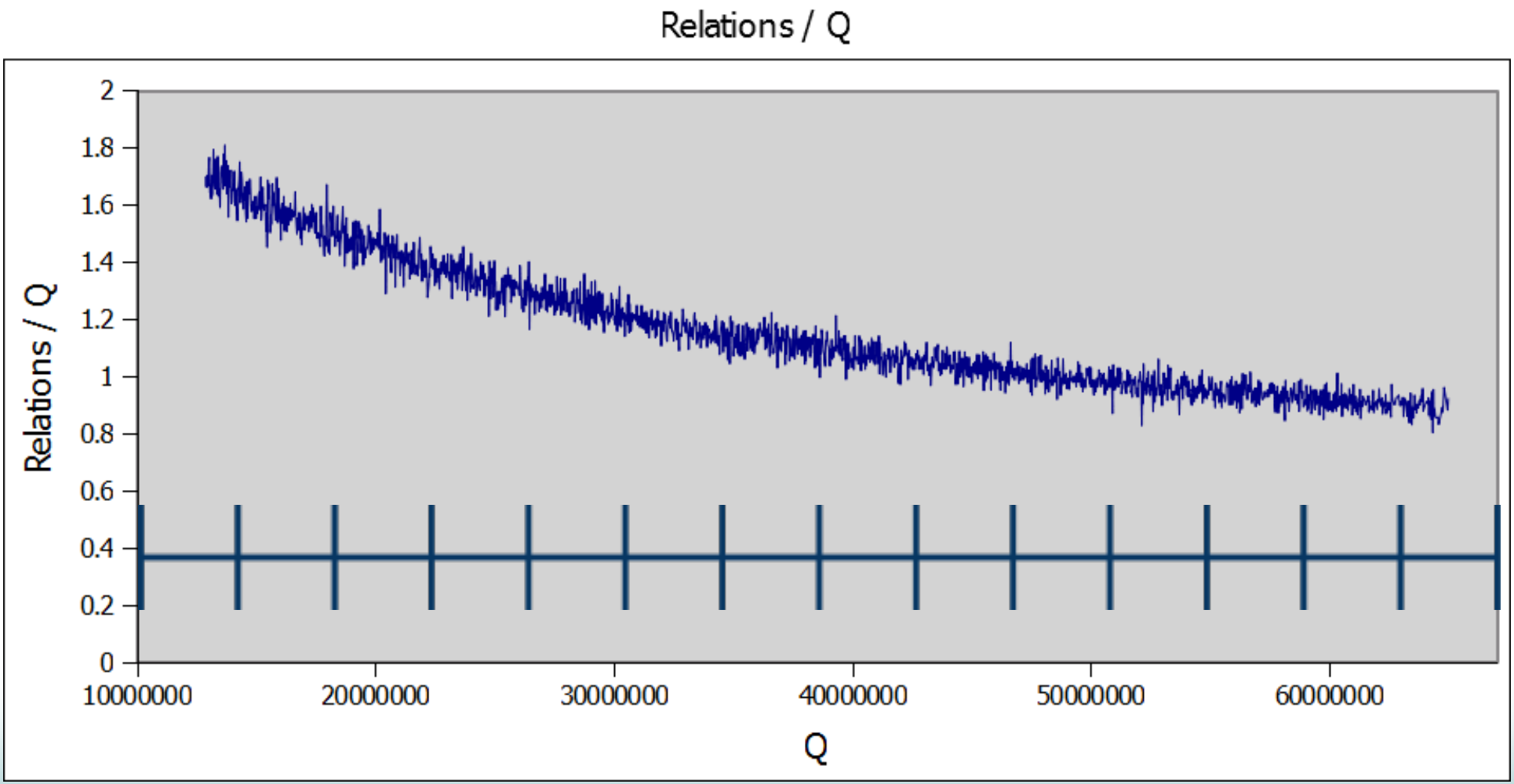


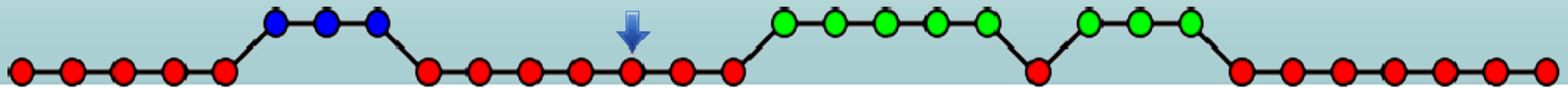
Sieving with GGNFS in BOINC

```
+#ifdef HAVE_BOINC
+int boincstart(int argc_init, char **argv) {
+  boinc_init();
+  boinc_resolve_filename("input_data", path_in, sizeof(path_in) );
+  boinc_resolve_filename("output_data", path_out,sizeof(path_out));
+  argv[argc_init++]="-R";
+  argv[argc_init++]="-a";
+  argv[argc_init++]="-o";
+  argv[argc_init++]=path_out;
+  argv[argc_init++]=path_in;
+  return argc_init;
+}
int main(int argc, char **argv) {
+       int app_argc, retcode;
+       char* app_argv[ARGVCOUNT];
+       app_argv[0] = argv[0];
+       app_argc= boincstart(1,app_argv);
+       retcode= main_lasieve(app_argc,app_argv);
+       boinc_finish(retcode);
+       return retcode;
+}
+int main_lasieve(int argc, char **argv)
+#else
+int main(int argc, char **argv)
+#endif
```

How Do I Factor

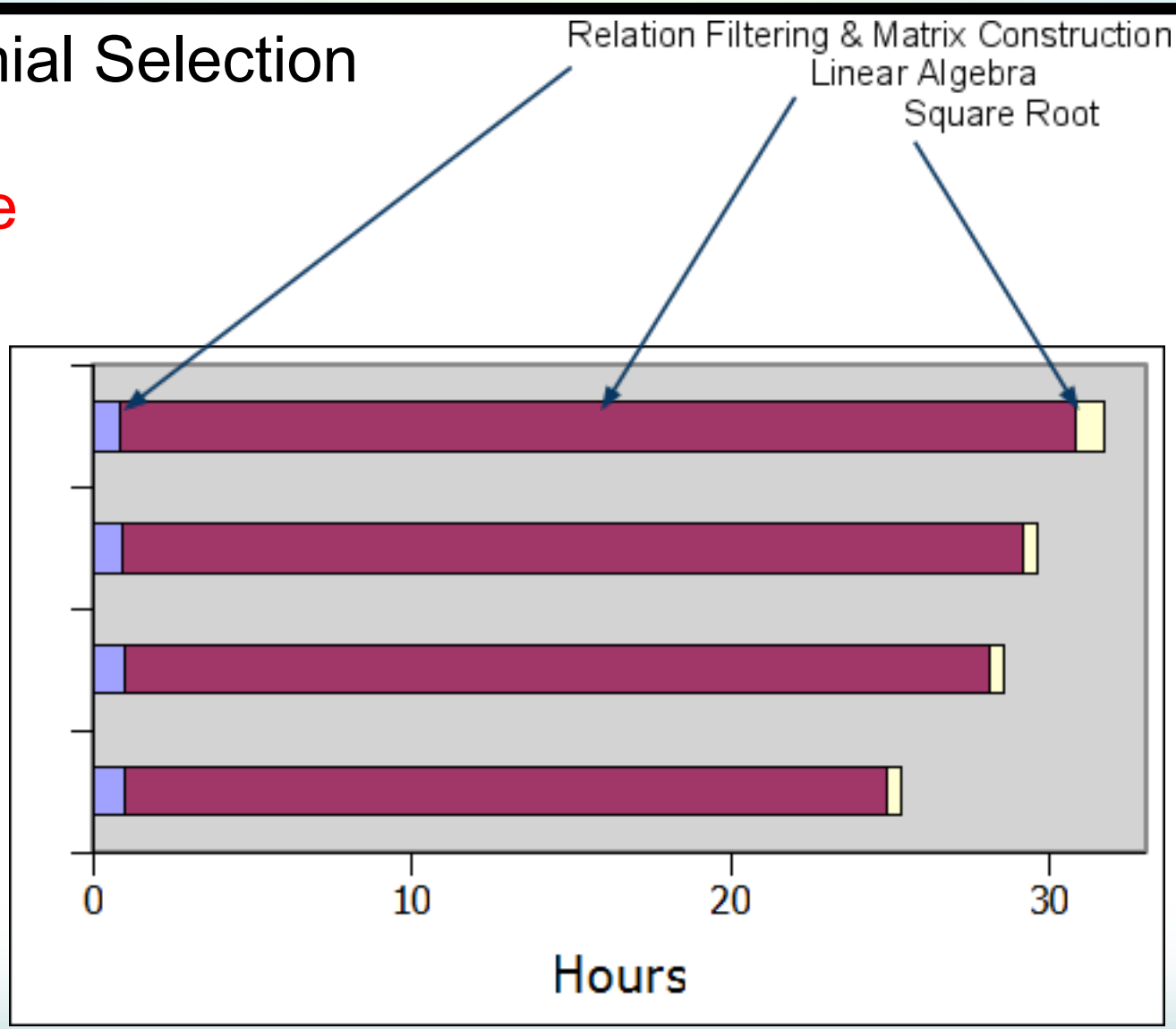
1. Polynomial Selection
2. Sieving

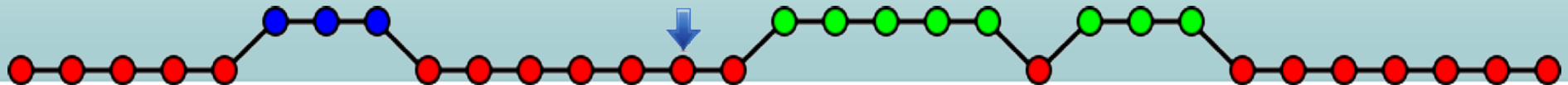




How Do I Factor

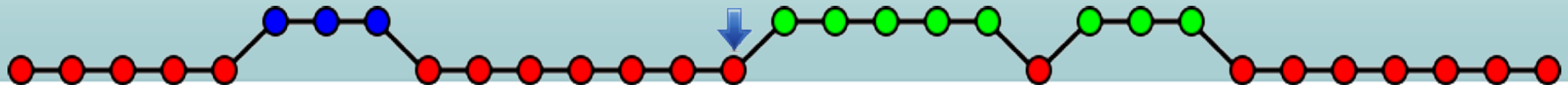
1. Polynomial Selection
2. Sieving
3. Combine



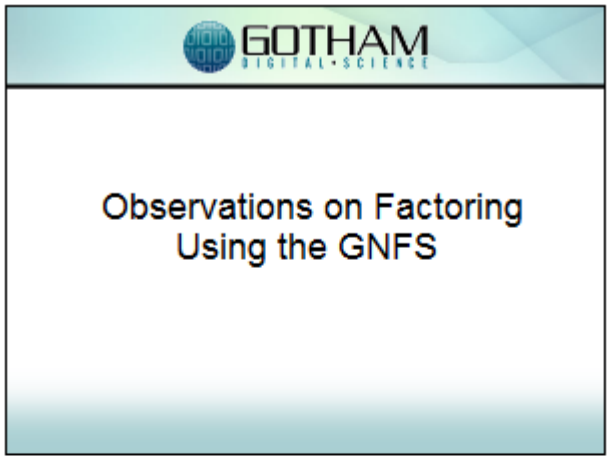


The payoff

```
$ wget -q https://www.eff.org/files/syrian-facebook-attack.pem
$ openssl x509 -noout -modulus -in syrian-facebook-attack.pem
Modulus=D5997DCA6577FCD964FE316987BDED93BA4D9644844629CF26CDA9CC
        EED253AD2EE646EE1CF8AC95D18FA014A2EC29672009BD684F79579A
        AA8D7E73E797F6B3
$ python
>>> n = int('D5997DCA6577FCD964FE316987BDED93BA4D9644844629CF26C
            DA9CCEED253AD2EE646EE1CF8AC95D18FA014A2EC29672009BD
            684F79579AAA8D7E73E797F6B3', 16)
>>> n
111871175171822184890047853438937107834419894175266549329387466
   591821609874883384428020723940086660859714316143876617034665
   78038190535215695710090863551231
>>> p = 10431832711621412355078235716253440775473942492929486918
        608964371166209731389 9
>>> q = 10724019284472797831715458754067770262540924005825331696
        4456831084693273770517 7
>>> n - (p * q)
0L
```

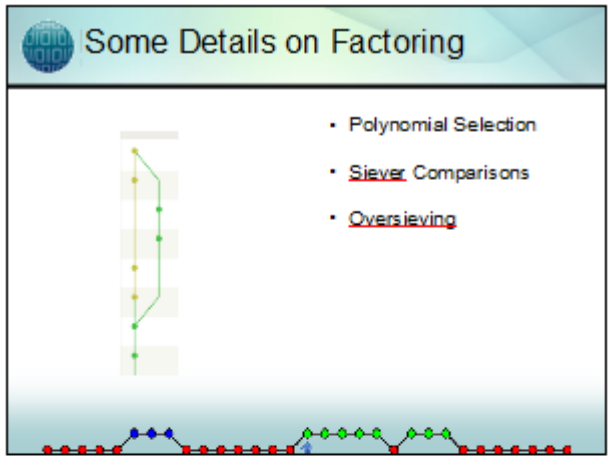


Factoring Details

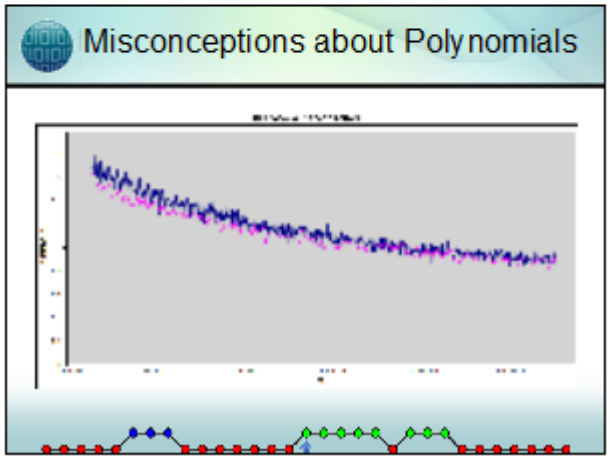


Cloud & Control

2

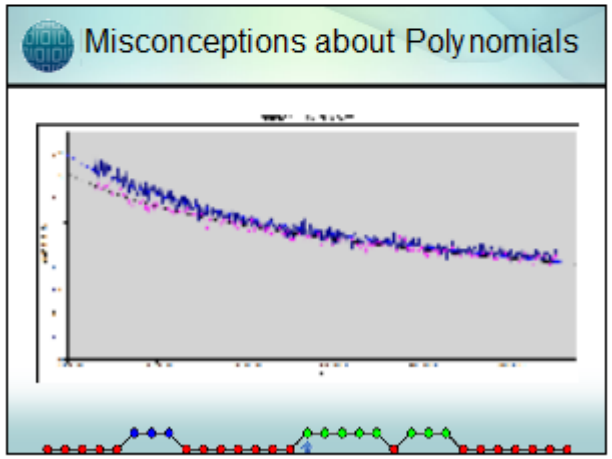


Some Details on Factoring



Misconceptions about Polynomials

4

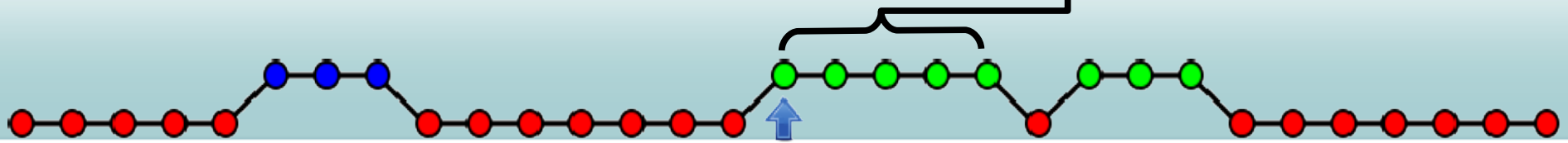


Slide 4

Moved to their own slide deck for time/relevance

Available on github.

(Sorry!)

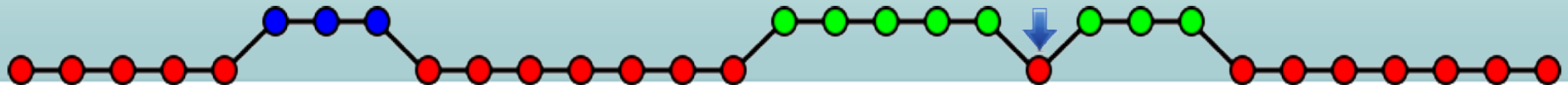


So Far

- BOINC
  - Why and How
  - Applications - Open source ⇨ BOINC Application
- Factoring RSA

Next

- BOINC
  - Close Source Applications
- Hands-off Cracking Passwords

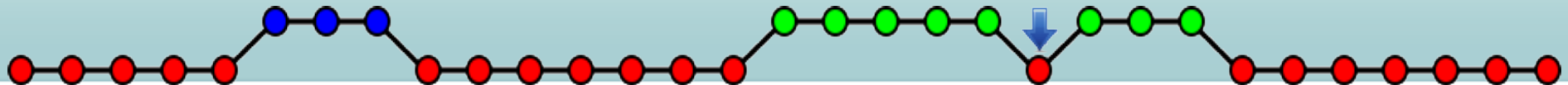


Cracking

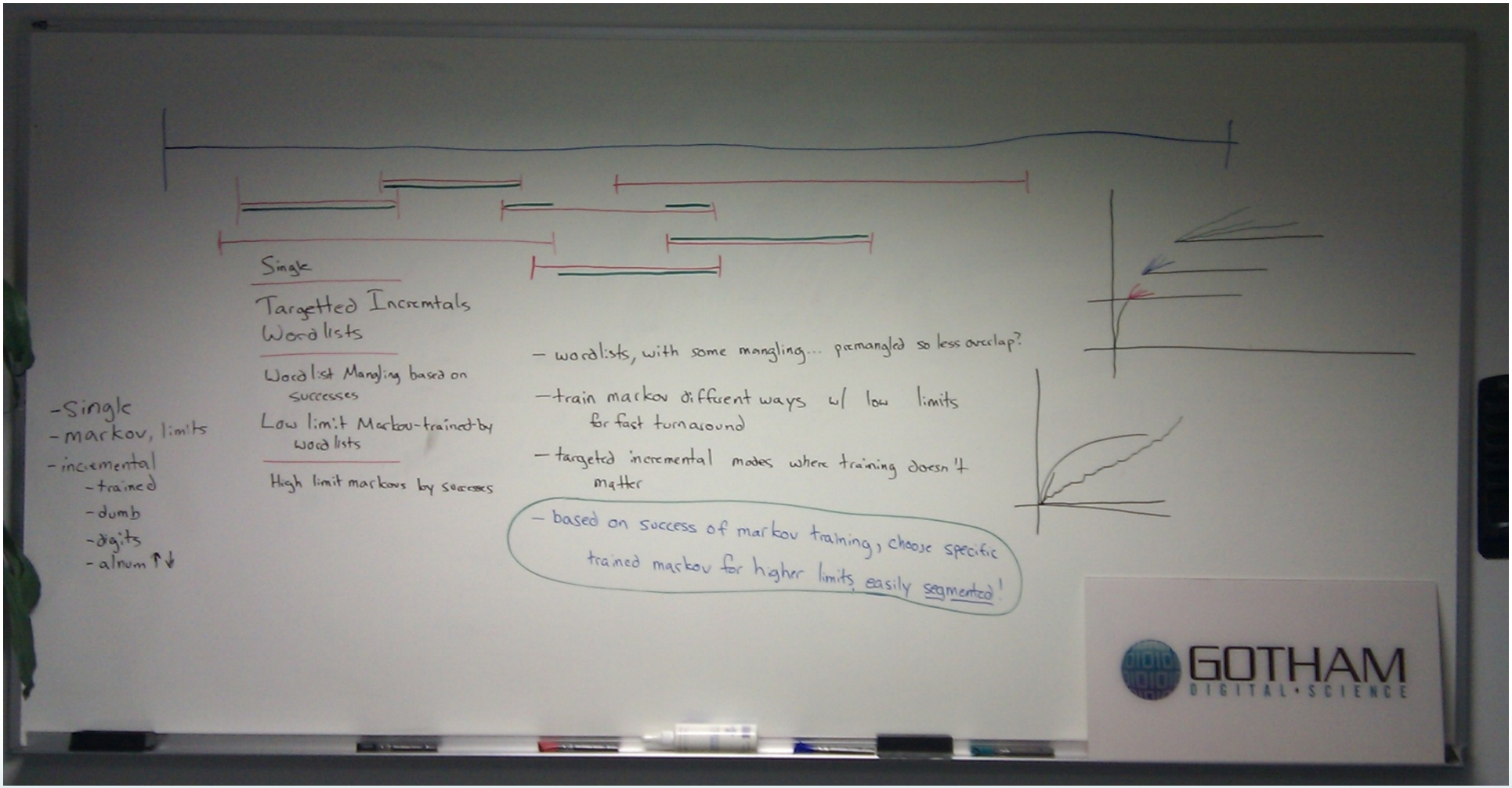
How do you Parrallelize Cracking?

- john
  - Several MPI Patches for john - but only on clusters
  - Mode:External - but non-trivial overhead when splitting
  - Cheap Hacks (bad idea)

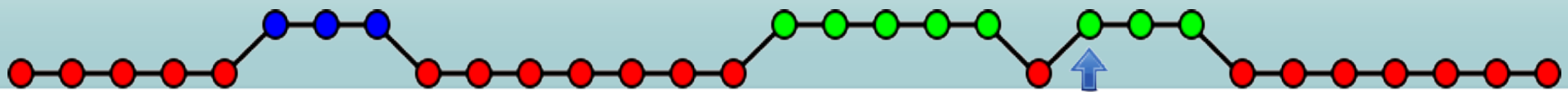
- hashcat family (hashcat, oclHashcat, cudaHashcat)
  - Not much you can do



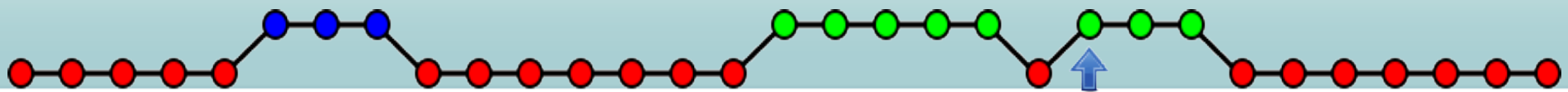
Enter the Magic



All Possible Passwords

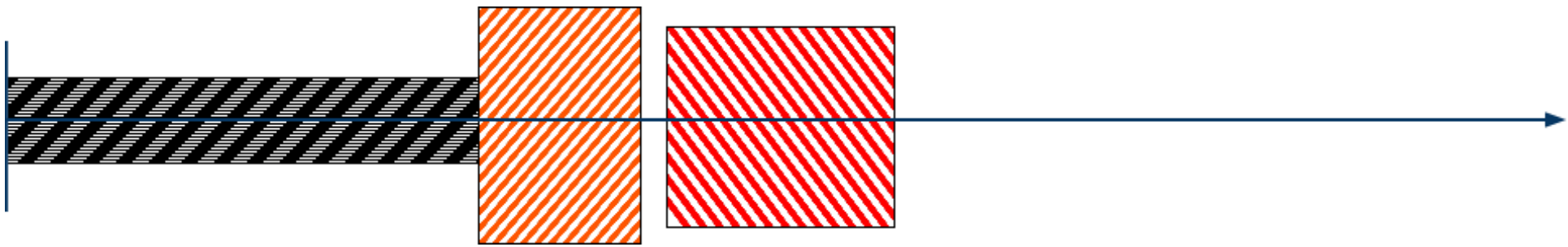


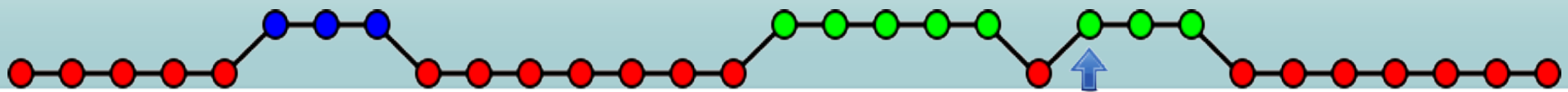
Brute Force



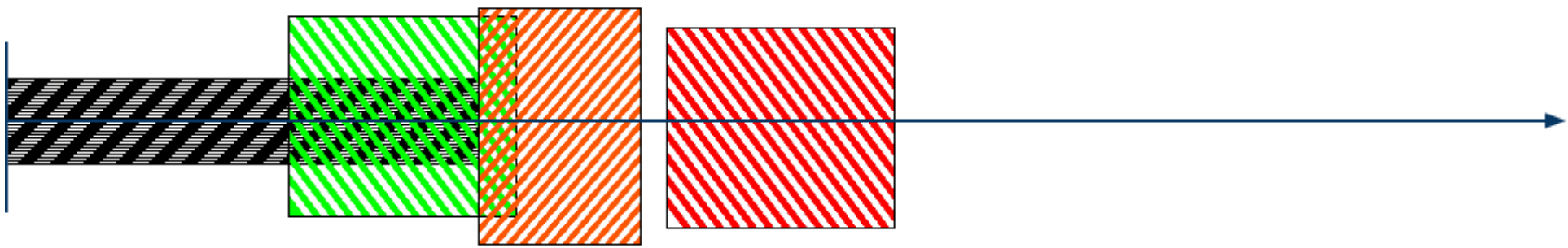
Wordlist

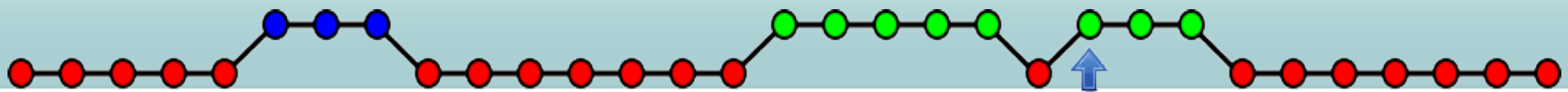
Two Wordlists!



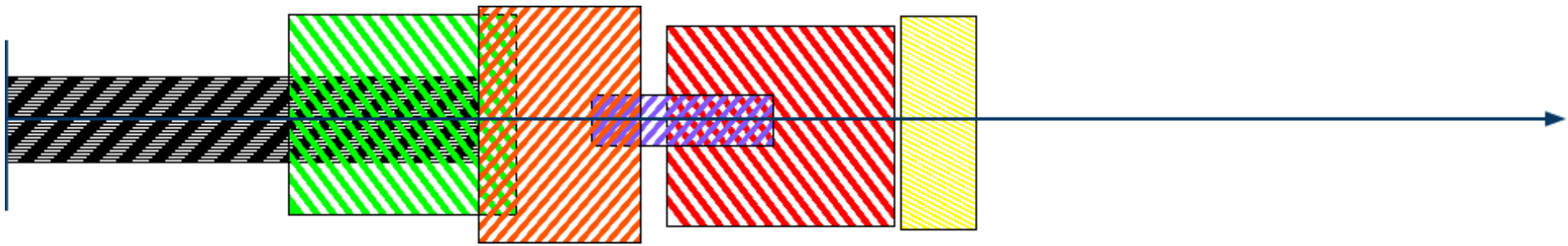


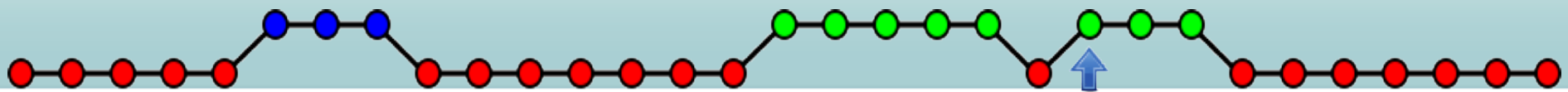
Let me try this....



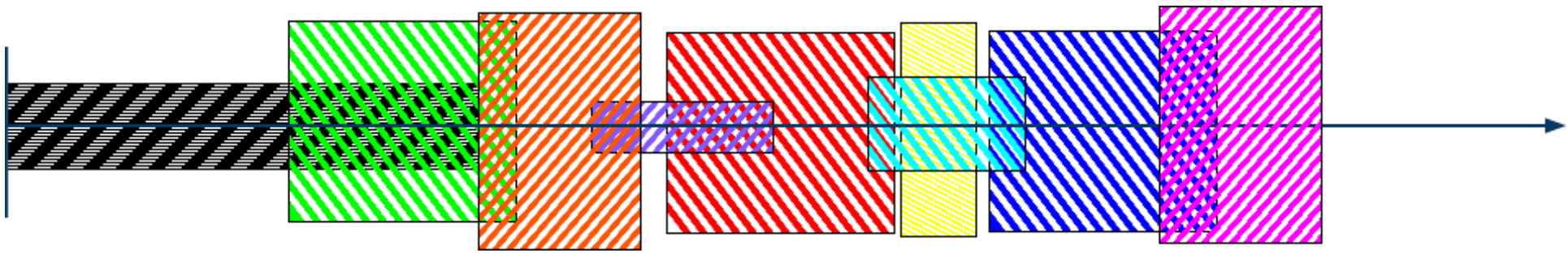


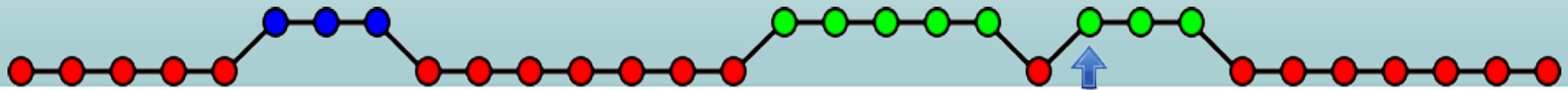
Foreign Wordlists!



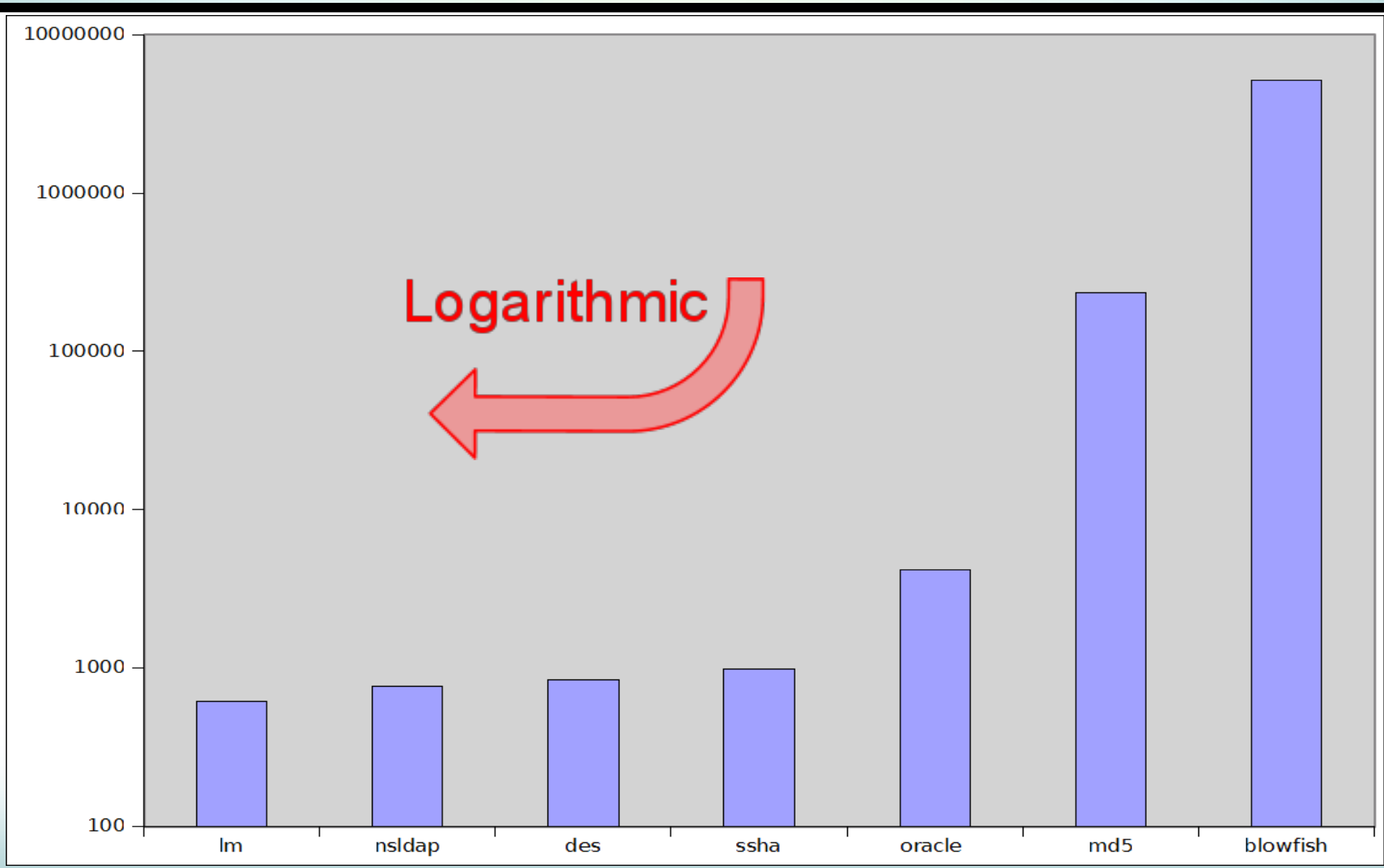


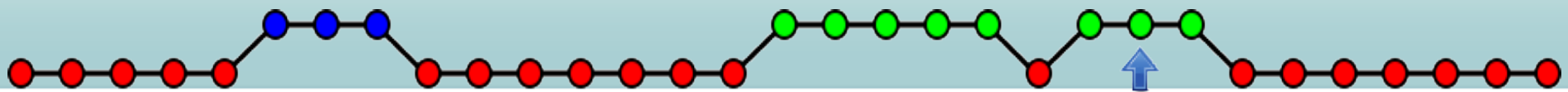
Kitchen Sink!





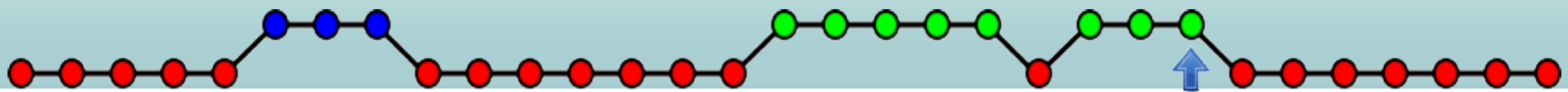
Not all hashes are created equal





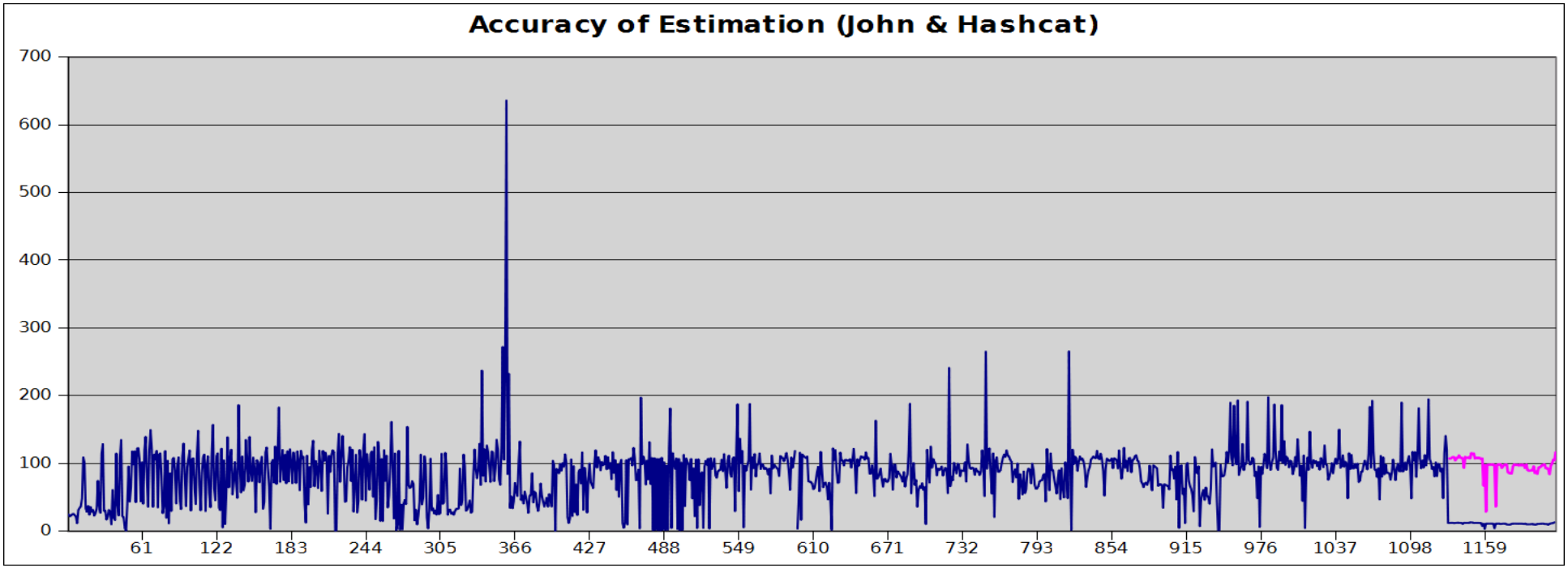
My Approach

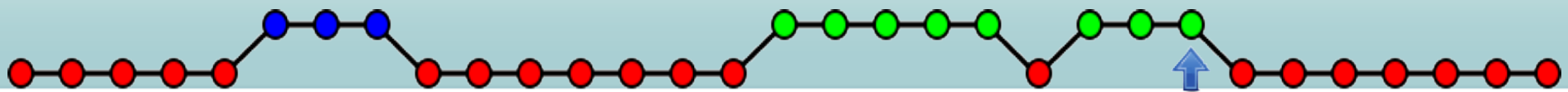
Phase 1: --single
Phase 2: 1 hour incremental



My Approach

Phase 1: --single
Phase 2: 1 hour incremental





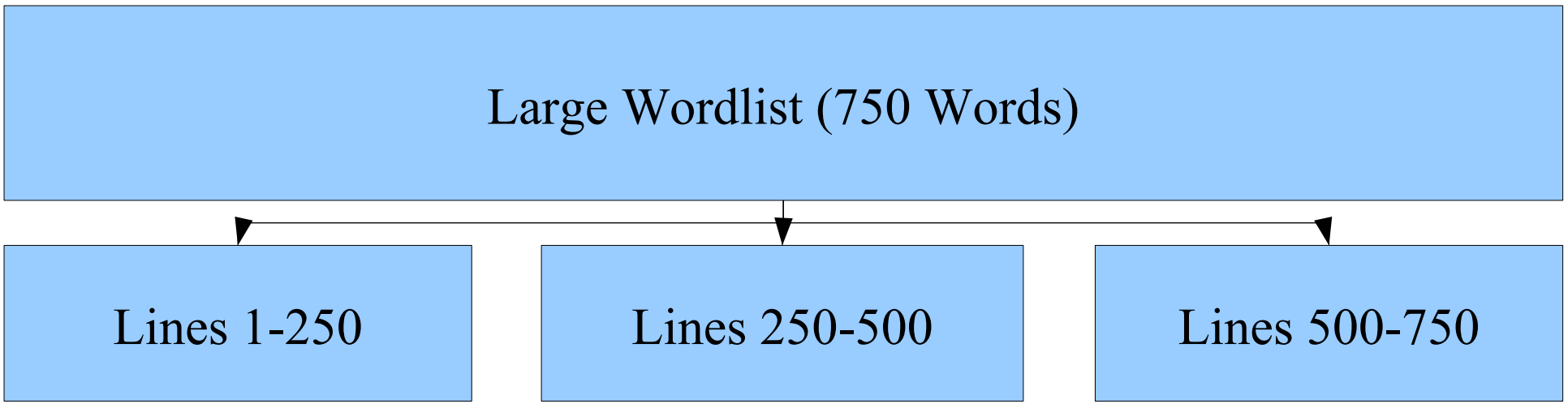
My Approach

Phase 1: --single
Phase 2: 1 hour incremental

Large Wordlist

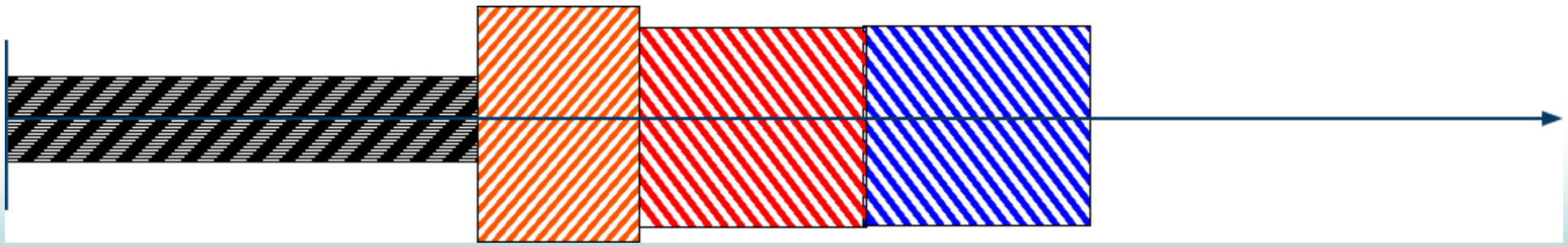
My Approach

Phase 1: --single
Phase 2: 1 hour incremental



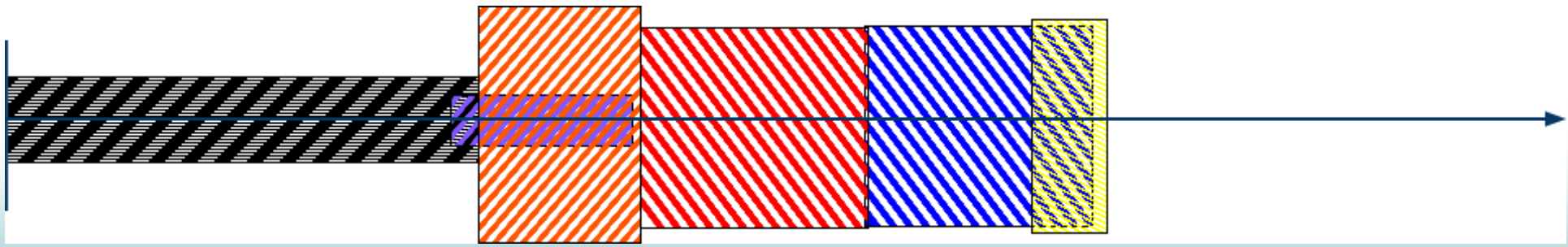
My Approach

Phase 1: --single
Phase 2: 1 hour incremental
Wordlists



My Approach

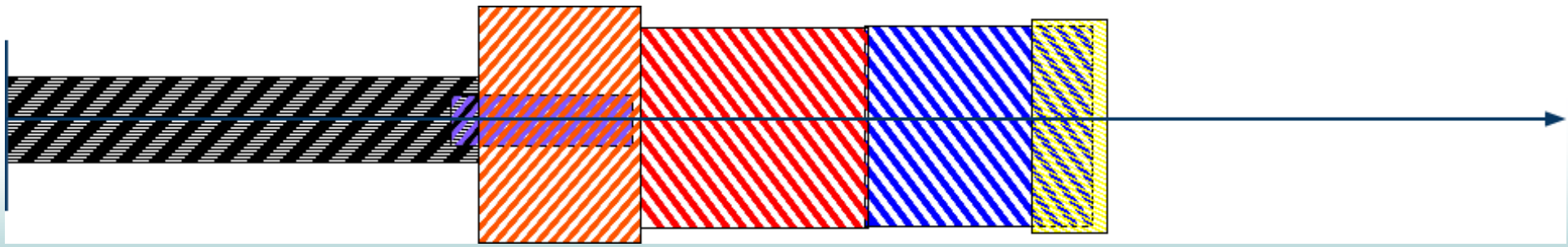
Phase 1: --single
Phase 2: 1 hour incremental
Wordlists
Phase 3: Wordlist Rules
High-Probability Markov Words

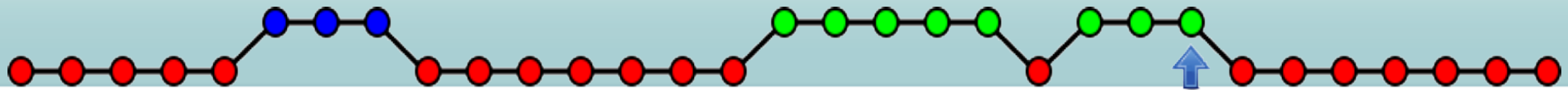


My Approach

Phase 1: --single
Phase 2: 1 hour incremental
Wordlists
Phase 3: Wordlist Rules
High-Probability Markov Words

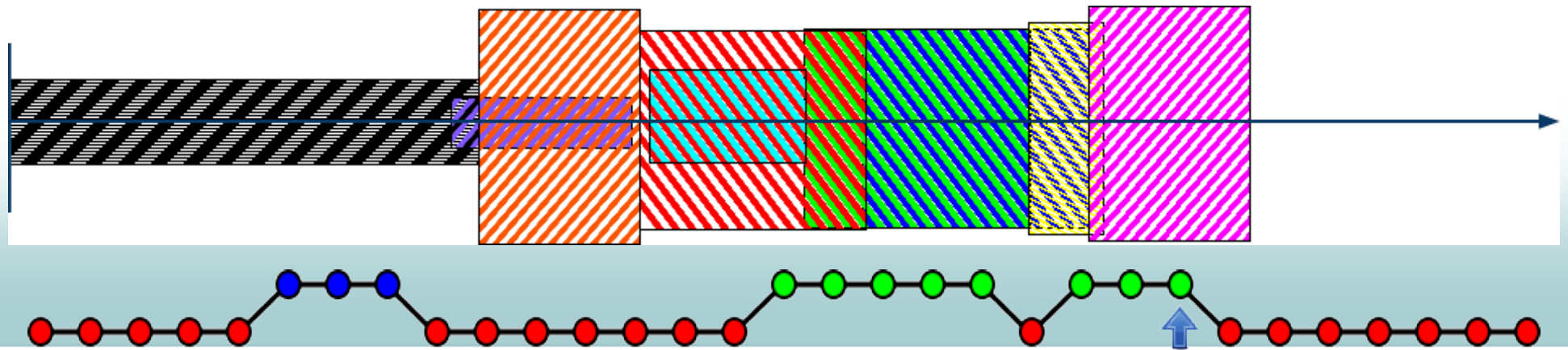
Very carefully pruned wordlists.





My Approach

Phase 1: --single
Phase 2: 1 hour incremental
Wordlists
Phase 3: Wordlist Rules
High-Probability Markov Words
Phase 4: Phase 3 Markovs + Rules
Low-Probability Markov Words
Phase 5: Phase 4 Markovs + Rules

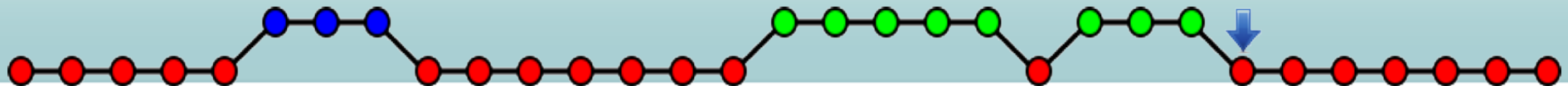


John the Ripper

Rewiring John into a BOINC App

```
+int main(int argc, char **argv) {
+       int status = boinc_init();
+       boinc_resolve_filename("john.conf", confFile, sizeof(confFile) );
+       boinc_resolve_filename("passwordlist", passlist, sizeof(passlist) );
+
+       int i, newArgc = 2, hasWordlist = 0;
+       for(i=1; i < argc; i++) {
+            newArgc++;
+            hasWordlist = strstr(argv[i], "<<WORDLIST>>") ? i : hasWordlist; }
+       if(hasWordlist) {
+           boinc_resolve_filename("wordlist", wordlistName, 512 );
+           snprintf(wordlistParameter, 612, "--wordlist=%s", wordlistName); }
+
+       newArgv[i=0] = argv[0];
+       for (i++; i<argc; i++) newArgv[i] = i == hasWordlist ? wordlistParameter : argv[i];
+       newArgv[i] = passlist;
+       int ret = john_main(newArgc, newArgv);
+       boinc_finish(ret);
+       return ret;
+}
+int john_main(int argc, char **argv)
+#else
int main(int argc, char **argv)
+#endif
```

*heavily abbreviated and trimmed



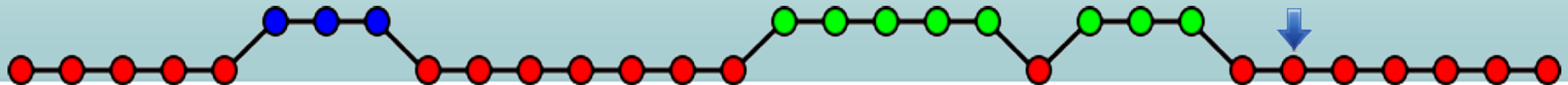
Application Versions

Add a new application:

1. Update project.xml
2. xadd

Add a new version:

1. copy files correctly
2. update_versions



Application Versions

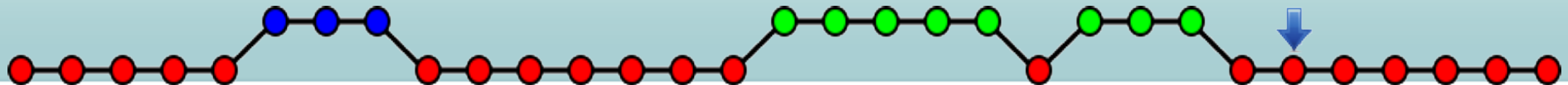
apps/
  name/
    name_version.minor_platform[.ext]

Add a new application:

1. Update project.xml
2. xadd

Add a new version:

1. copy files correctly
2. update_versions



Application Versions

apps/
  name/
    name_version.minor_platform[.ext]

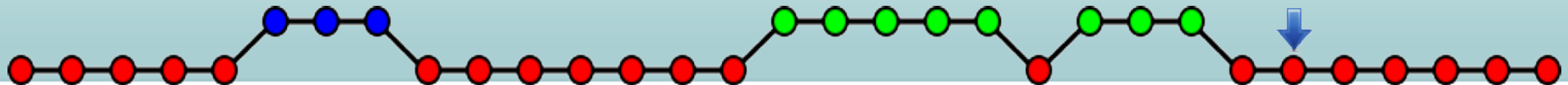
  msieve/
    msieve_148.1_linux

Add a new application:

1. Update project.xml
2. xadd

Add a new version:

1. copy files correctly
2. update_versions



Application Versions

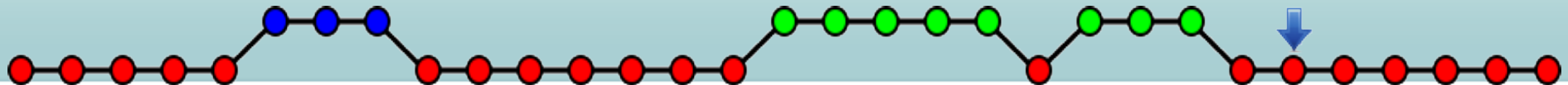
```
apps/
  name/
    name_version.minor_platform[.ext]

  msieve/
    msieve_148.1_linux


  newapp/
    newapp_1.0_linux/
      newapp_1.0_linux
      resourcefile.dat
      somethingelse.db
```

Add a new application:

1. Update project.xml
2. xadd

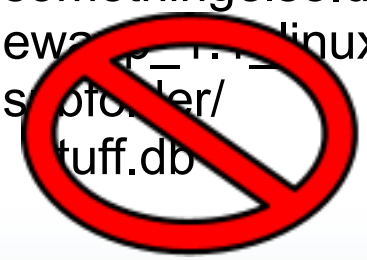
Add a new version:

1. copy files correctly
2. update_versions



Application Versions

```
apps/
  name/
    name_version.minor_platform[.ext]

  msieve/
    msieve_148.1_linux


  newapp/
    newapp_1.0_linux/
      newapp_1.0_linux
      resourcefile.dat
      somethingelse.db
    newa[illegible]linux/
      s[illegible]er/
        [illegible]tuff.db
```

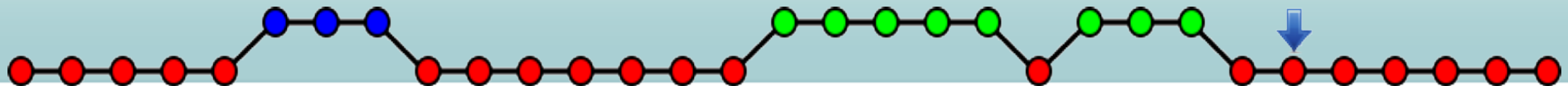


Add a new application:

1. Update project.xml
2. xadd

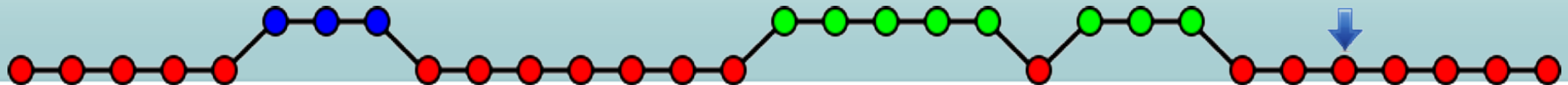
Add a new version:

1. copy files correctly
2. update_versions



Hashcat

hashcat, oclhashcat, oclhashcat+, oclhashcat-lite

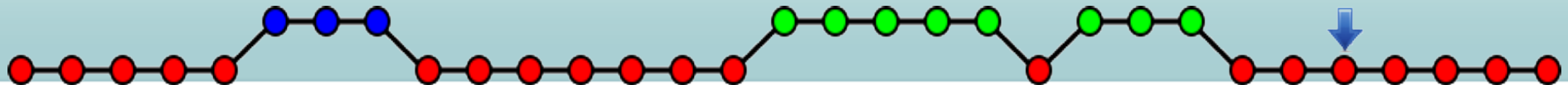


BOINC & Closed Source Apps: Wrapper Apps

job.xml

```
<job_desc>
  <task>
    <application>hashcat</application>
    [ <stdin_filename>name</...> ]
    [ <stdout_filename>name</...> ]
    [ <stderr_filename>name</...> ]
    [ <command_line>--foo bar</...> ]
    [ <append_cmdline_args/> ]
  </task>
  <task>
     ...
  </task>
</job_desc>
```

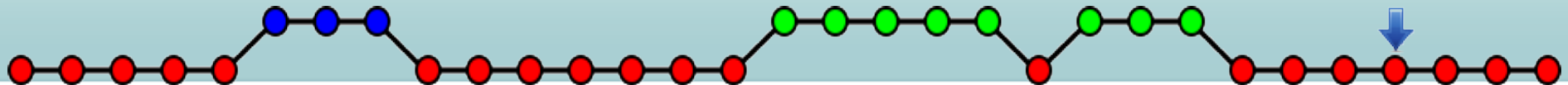
- Features!
  - <daemon />
  - <multi_process />
  - <setenv>
- genwrapper
  - functionally bash
  - for, while, if
  - cat, egrep, sed, awk, sort, gzip, unix2dos,...



App Plans & GPU Stuff

apps/
  name/
    name_version.minor_platform[.ext]

  msieve/
    msieve_148.1_linux

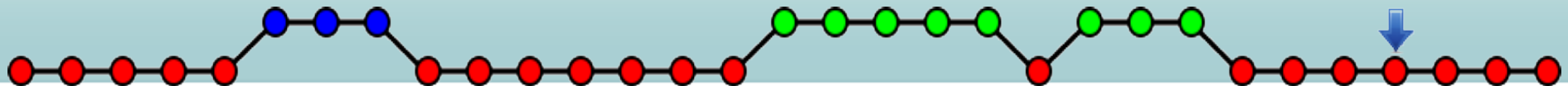


App Plans & GPU Stuff

apps/
  name/
    name_version.minor_platform[.ext]

  msieve/
    msieve_148.1_linux

  cudahashcat+/
    cudahashcat+_3.1_linux__cuda



App Plans & GPU Stuff

apps/
  name/
    name_version.minor_platform[.ext]

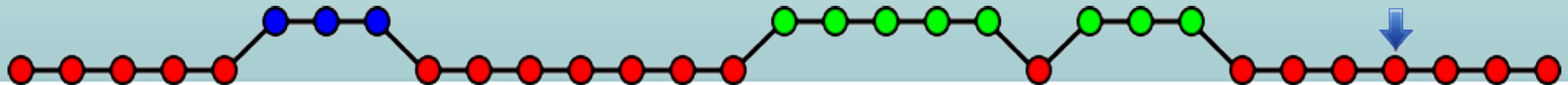
  msieve/
    msieve_148.1_linux

  cudahashcat+/
    cudahashcat+_3.1_linux__cuda

__mt - Multi-threaded
__cuda

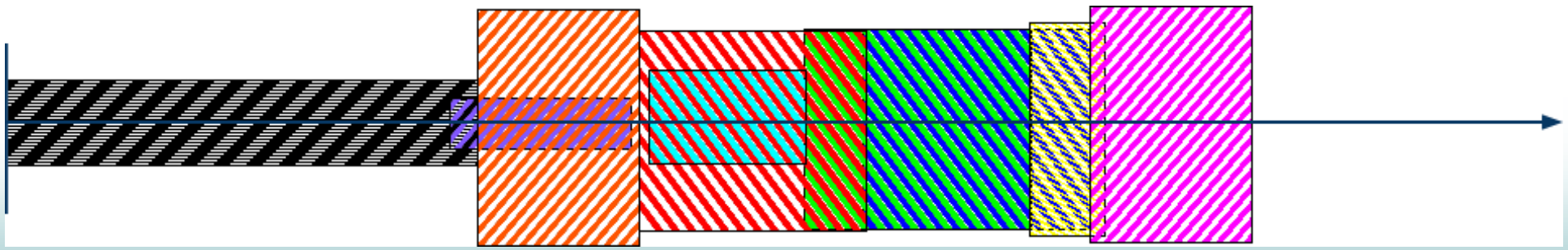
Specific GPU Targets:
    __cuda_fermi
    __cuda_opencl
    __ati14
    ...

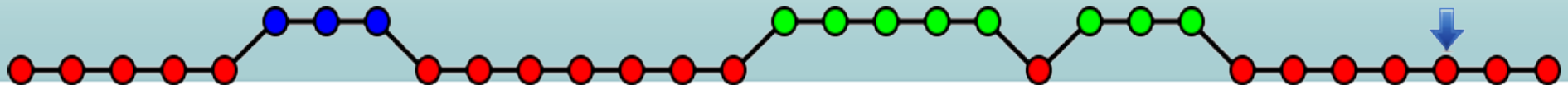
__nci - Non-CPU Intensive
__sse3
__vbox32 - VirtualBox



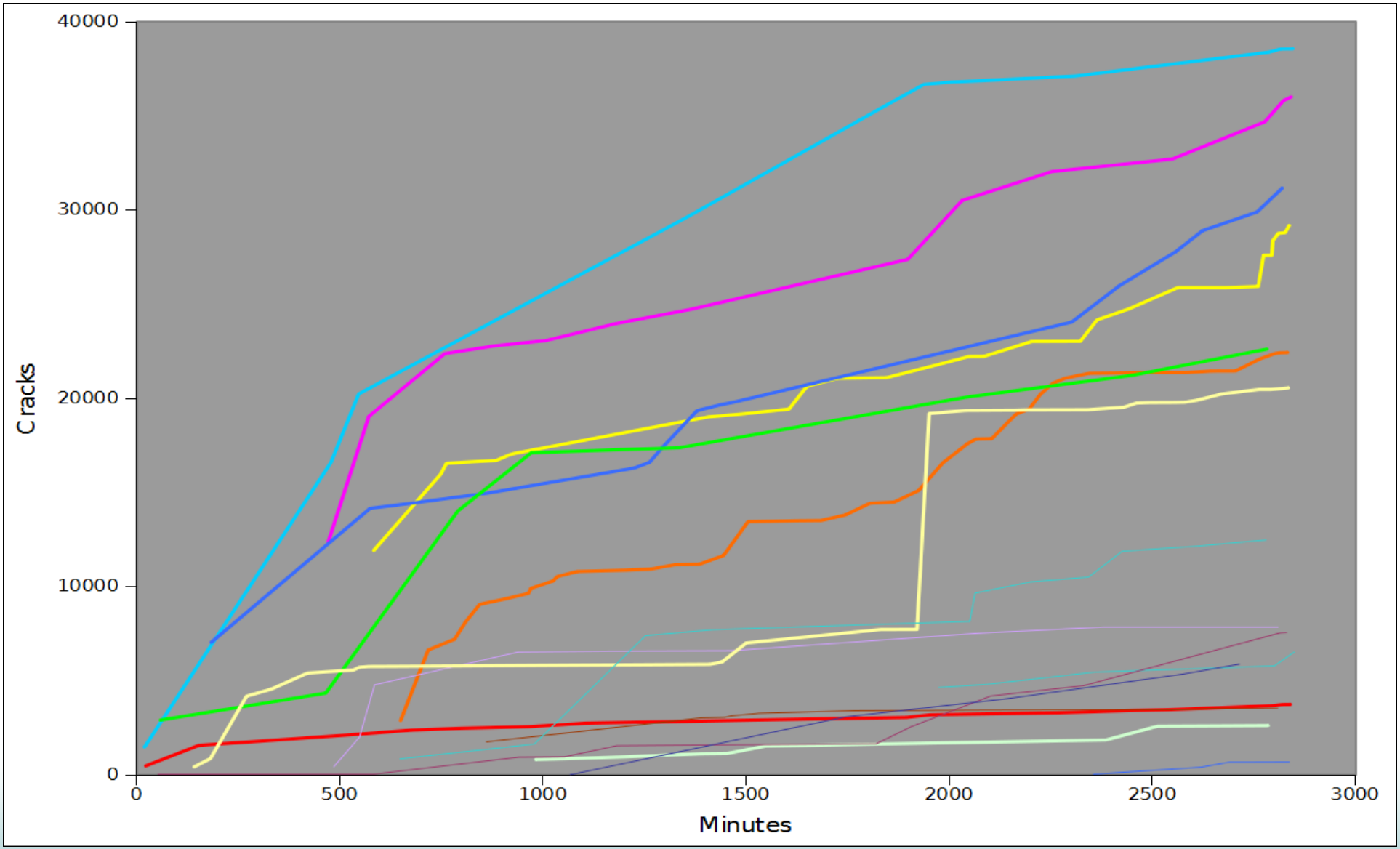
My Approach

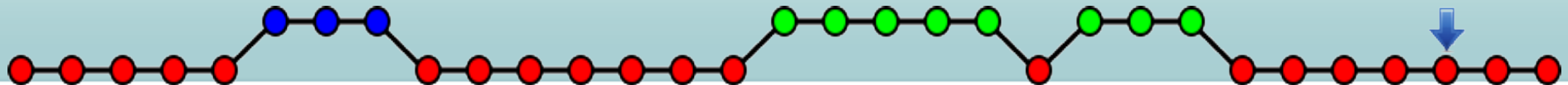
Phase 1: --single
Phase 2: 1 hour incremental
Wordlists
Phase 3: Wordlist Rules
High-Probability Markov Words
Phase 4: Phase 3 Markovs + Rules
Low-Probability Markov Words
Phase 5: Phase 4 Markovs + Rules



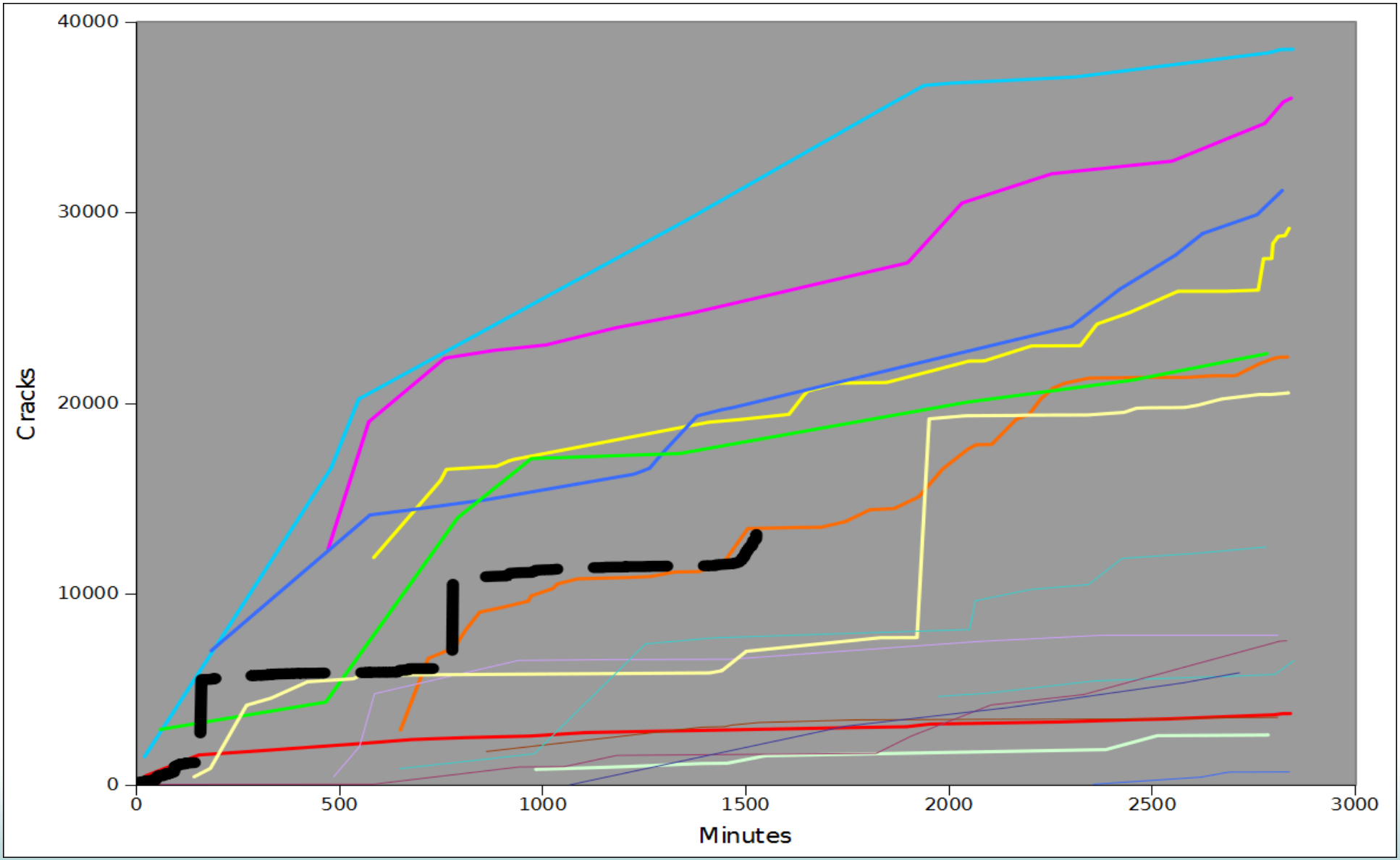


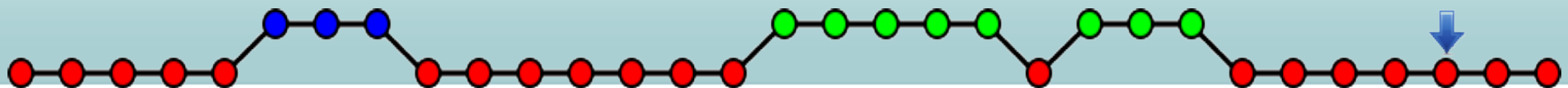
Benchmarking: 2010 Defcon Korelogic Crack Me If You Can Contest



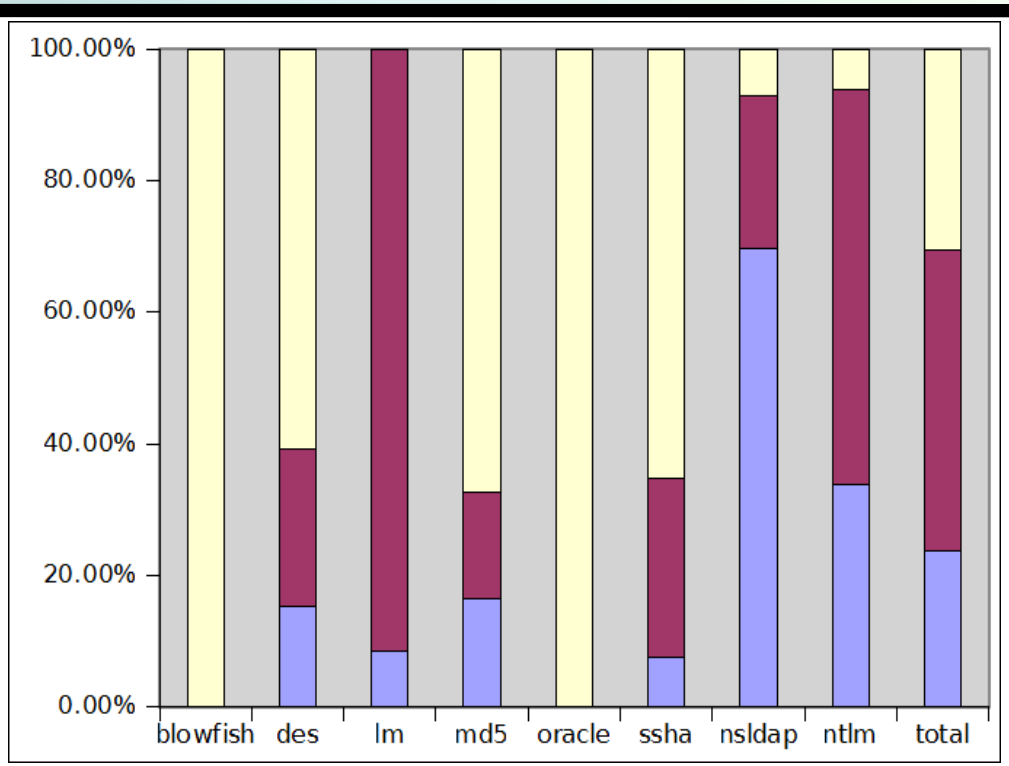


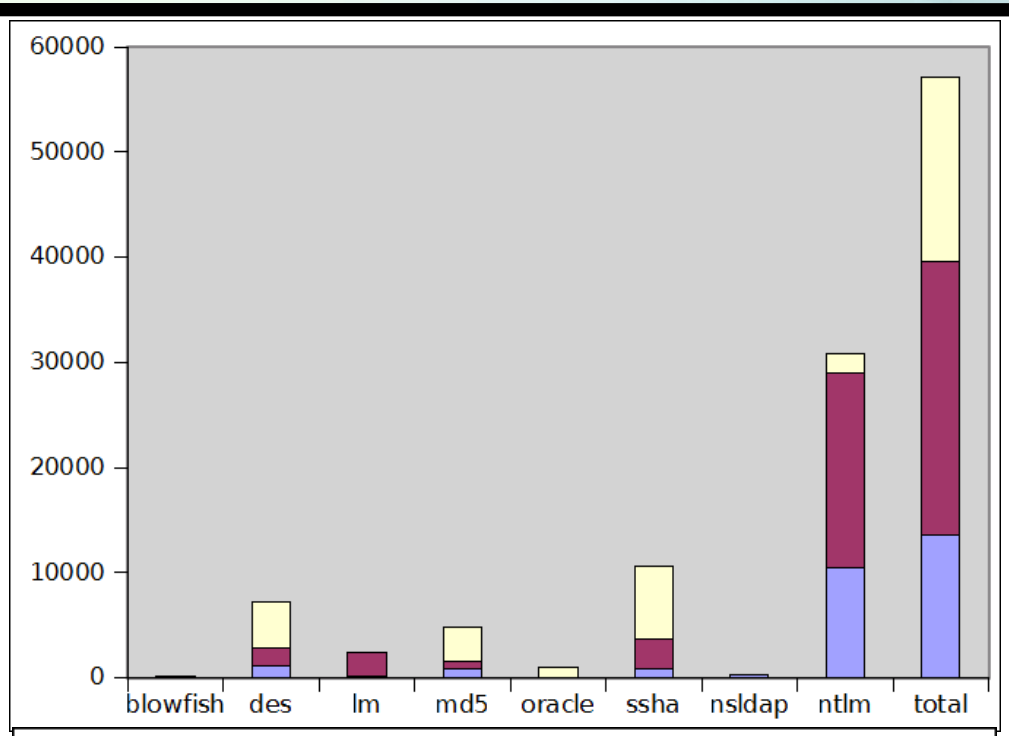
Abject Failure.

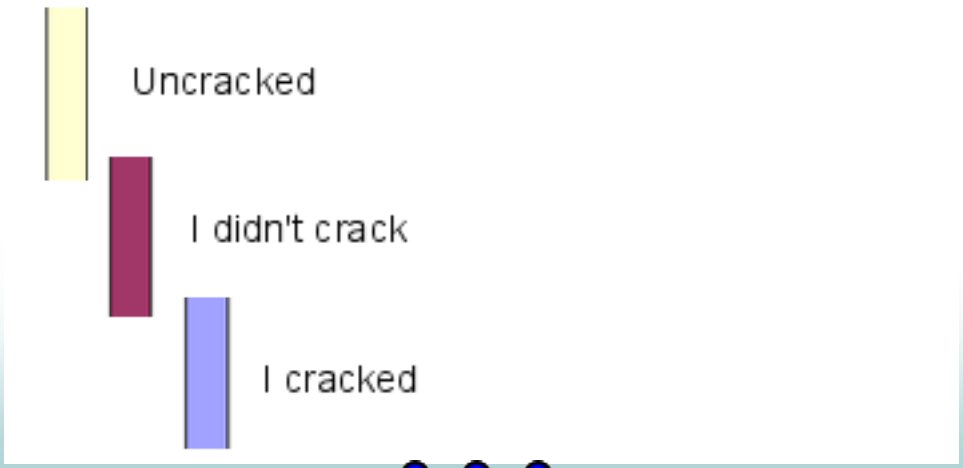


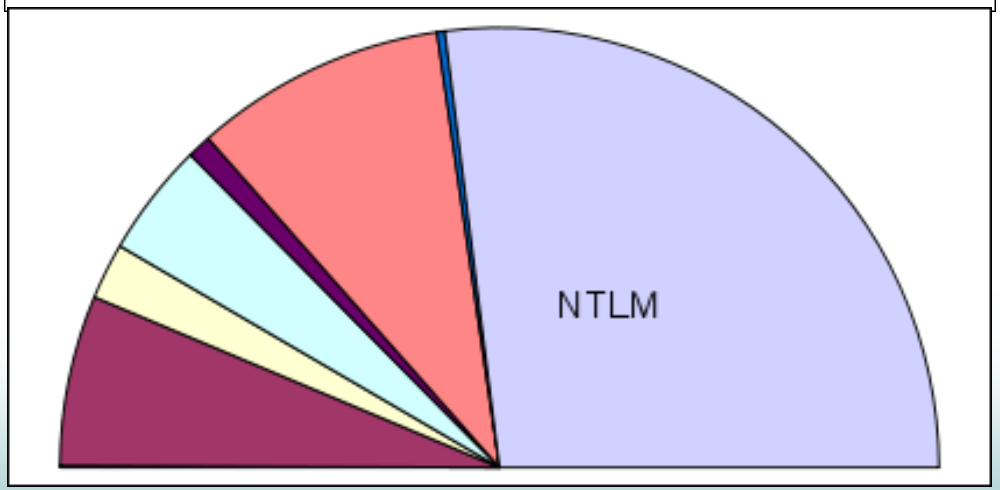


Failure by Hash Type









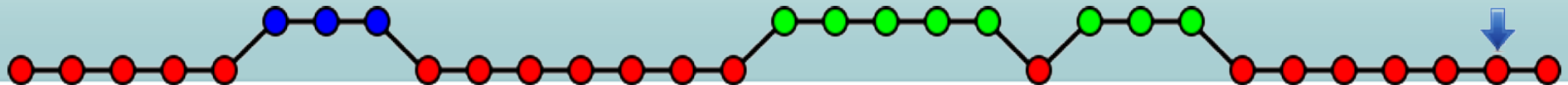
Lessons Learned

- Iterative Cracking

```
    new pattern -> maskprocessor -> rules -> cracks
         /\                                   \/
      new plains   <-   random rules   <-   new dic
```

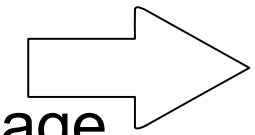
- Automatic mangle rules creation
- Observations from cracked passwords

- Cracked password lists
- Actually Crack LM



Back to BOINC

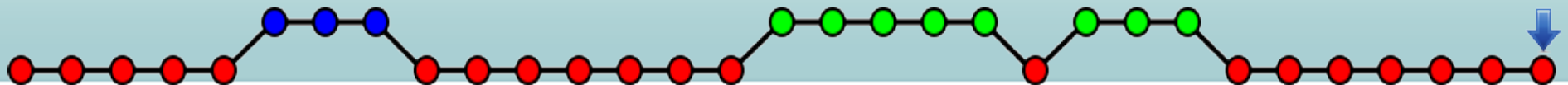
1. Set up a BOINC Server
2. Edit config.xml
3. Lock down the server
4. Set up a client image
5. Set up an application
6. Automate the client image
7. ???
8. Profit!



1. Patch source

   or

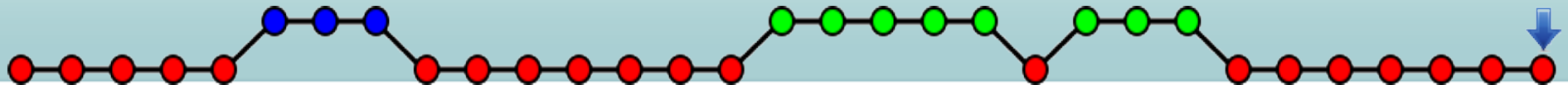
1. Write job.xml
2. Write input & output templates
3. update_versions
4. Create test workunits
5. Test
6. Repeat 1-6 as needed



Alternatives to BOINC

Password Cracking Only
- Browser Based using Javascript / AJAX / Web Workers
- Durandal http://durandal-project.org/
- Rick Redman of Korelogic's tool

General Architecture
- Amazon Elastic Beanstalk (Java-only)
- Amazon SQS (Write your own wrapper and uploader)
- Bash Scripts/tentakel/multixterm/cssh
- Write your own?
  Will that take more or less than time than configuring BOINC?
  I think more.



Obligatory Ending Slide

Questions?

Thanks:

- GDS
- NYSec
- MersenneForum & jasonp

Tom Ritter
http://ritter.vg
(encrypted mail preferred)

Big Ups To:

- jasonp
- BOINC Devs

http://www.gdssecurity.com/ Hiring: NYC & London
https://github.com/GDSSecurity/cloud-and-control