Announcements

- **Lab 2 now on web site**
- **Next week my office hours moved to Monday 4:30pm**
- **This week office hours Wednesday 4:30pm as usual**
- **Weighting of papers for final discussion**

[discussion of listen]

Bro: Detecting network intruders

- **Many security holes exploited over the network**
  - Buffer overruns in servers
  - Servers with bad implementations
    ("login -froot", telnet w. LD_LIBRARY_PATH)
- **Goal: Detect people exploiting such bugs**
- **Detect activities performed by people who've penetrated server**
  - Setting up IRC bot
  - Running particular commands, etc.

Bro model

- **Attach machine running Bro to "DMZ"**
  - Demilitarized zone – area betw. firewall & outside world
- **Sniff all packets in and out of the network**
- **Process packets to identify possible intruders**
  - Secret, per-network rules identify possible attacks
  - Is it a good idea to keep rules secret?
- **React to any threats**
  - Alert administrators of problems in real time
  - Switch on logging to enable later analysis of potential attack
  - Take action against attackers – E.g., filter all packets from host that seems to be attacking

Goals of system

- **Keep up with high-speed network**
  - No packet drops
- **Real-time notification**
- **Separate mechanism from policy**
  - Avoid easy mistakes in policy specification
  - So different sites can specify “secret” policies easily
- **Extensibility**
- **Resilience to attack**

Challenges

- **Have to keep up with fast packet rate**
- **System has to be easy to program**
  - Every site needs different, secret rules
  - Don't want system administrators making mistakes
- **Overload attacks**
- **Crash attacks**
- **Subterfuge attacks**

Bro architecture

- **Layered architecture:**
  - bpf/libpcap, Event Engine, Policy Script Interpreter
- **Lowest level bpf filter in kernel**
  - Match interesting ports or SYN/FIN/RST packets
  - Match IP fragments
  - Other packets do not get forwarded to higher levels
- **Event engine, written in C++**
  - Knows how to parse particular network protocols
  - Has per-protocol notion of events
- **Policy Script Interpreter**
  - Bro language designed to avoid easy errors

Overload and Crash attacks

- **Overload goal: prevent monitor from keeping up w. data stream**
  - Leave exact thresholds secret
  - Shed load (e.g., HTTP packets)
- **Crash goal: put monitor out of commission**
  - E.g., run it out of space (too much state)
  - Watchdog timer kills & restarts stuck monitor
  - Also starts tcpdump log, so same crash attack, if repeated, can be analyzed

Challenges

- **Dealing with FTP**
  - Separate pipelined requests
  - Parse PORT command to detect "bounce" attacks
- **Dealing with type-ahead and rejected logins with telnet/rlogin**
  - Flows basically unstructured–don't know what's username
  - Use heuristics (e.g., look for "Password:" string)
  - But typeahead makes it harder to match exactly
- **Network scans and port scans…How to detect**
  - Keep table of connection attempts (src, dst, bool)
  - If not seen yet, increment count of distinct_peers[src]
  - Trade-off between state recovery & detection of slow scans

Subterfuge attacks

- **IP fragments too small to see TCP header**
- **Retransmitted IP fragments w. different data**
- **Retransmitted TCP packets w. different data**
- **TTL/MTU monkeying can hide packets from destination**
  - Compare TCP packet to retransmitted copy
  - Assume one of two endpoints is honest (exploit ACKs)
  - Bifurcating analysis

State and checkpointing

- **Need to keep a lot of session state**
  - Open TCP connections, UDP request-response, IP fragments
  - No timers to garbage collect state
- **Checkpointing the system**
  - Start new copy of monitoring process
  - Kill old copy when new copy has come up to speed
  - Is this ideal?

Quarantining worms

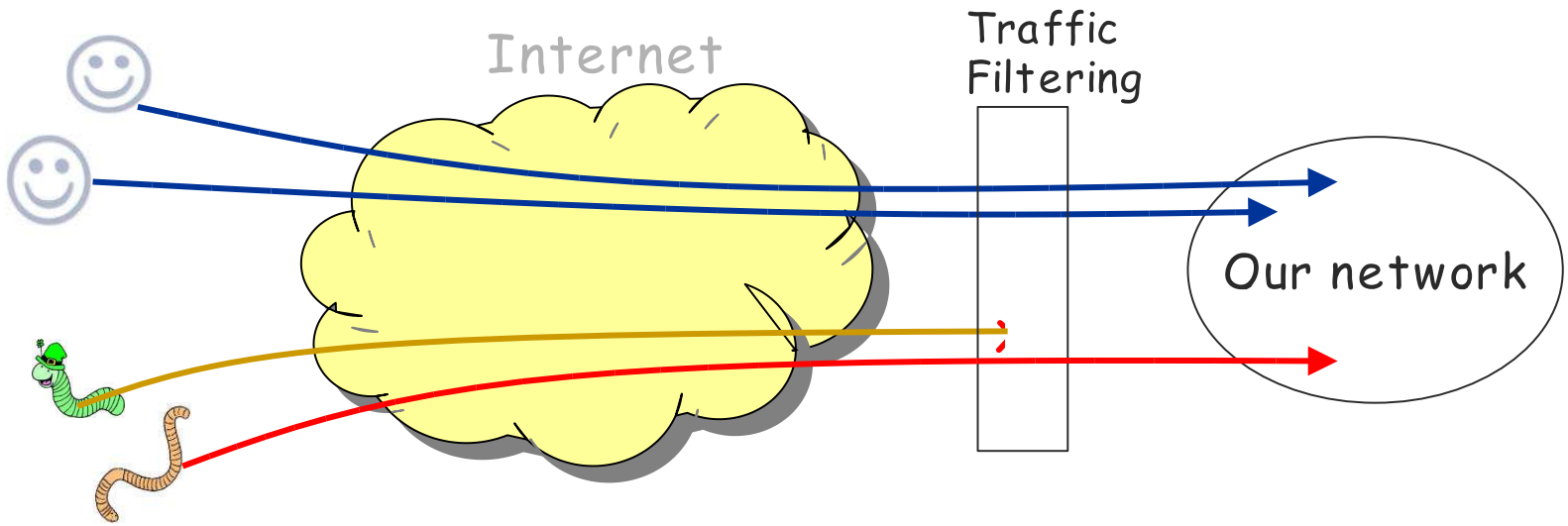
- **Several possible techniques**
  - Destination port blocking
  - Infected source host IP blocking
  - Content-based blocking
- **Key idea: Worms have signatures**

```
05:45:3                                  > 209.78.235d.128.80: . 0:1460(1460)
ack 1 w
0x0000   4500 05dc 84af 4000 6f06 5315 5ac4 16c4 E.....@.o.S.Z...
0x0010   d14e eb80 06b4 0050 5e86 fe57 440b 7c3b .N.....P^..WD.|;
0x0020   5010 2238 6c8f 0000 4745 5420 2f64 6566 P."8l...GET./def
0x0030   6175 6c74 2e69 6461 3f58 5858 5858 5858 ault.ida?XXXXXXX
0x0040   5858 5858 5858 5858 5858 5858 5858 5858 XXXXXXXXXXXXXXXX
0x
0x
0x
0x0110   5858 5858 5858 5858 5825 7539 3039 3025 XXXXXXXX%u9090%
0x01a0   303d 6120 4854 5450 2f31 2e30 0d0a 436f 0=a.HTTP/1.0..Co
```

Signature for CodeRed II

Signature: A Payload Content String Specific To A Worm

Use signatures for Content-based blocking



- **Combine with Bro, or other similar tools (Snort, NBAR)**

Problem: Signature derivation too slow

- **Current Signature Derivation Process**
  - New worm outbreak
  - Report of anomalies from people via phone/email/newsgroup
  - Worm trace is captured
  - Manual analysis by security experts
  - Signature generation
- **Labor-intensive, Human-mediated**

Autograph goal

- **Automatically generate signatures of previously unknown Internet worms**
  - as accurately as possible
    $\Longrightarrow$ Content-Based Analysis
  - as quickly as possible
    $\Longrightarrow$ Automation, Distributed

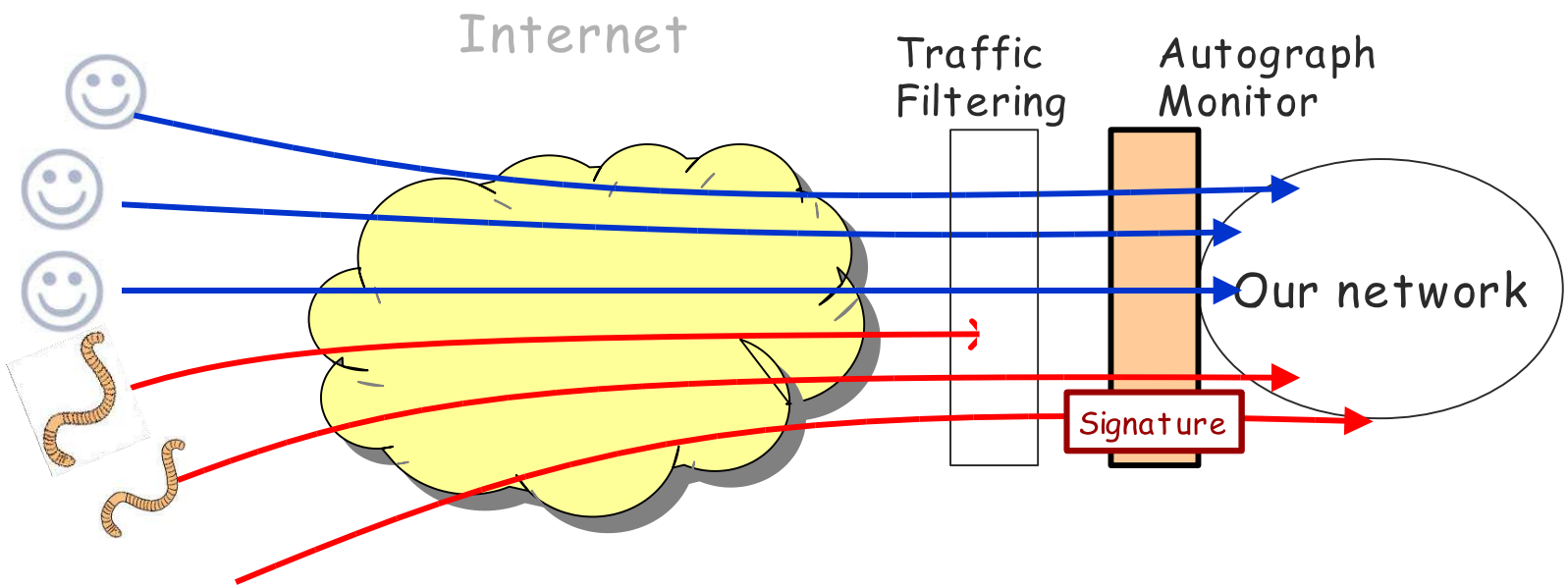
Automation assumptions

- **Focus on TCP worms that propagate via scanning**
- **Actually, any transport**
  - in which spoofed sources cannot communicate successfully
  - in which monitor knows transport framing
- **Worm's payloads share a common substring**
  - Vulnerability exploit part is not easily mutable
  - No polymorphic worms [for now]

Success metrics

- **Automation: Minimal manual intervention**
- **Signature quality: Sensitive & specific**
  - Sensitive: match all worms low false negative rate
  - Specific: match only worms low false positive rate
- **Timeliness: Early detection**
- **Application neutrality (Broad applicability)**

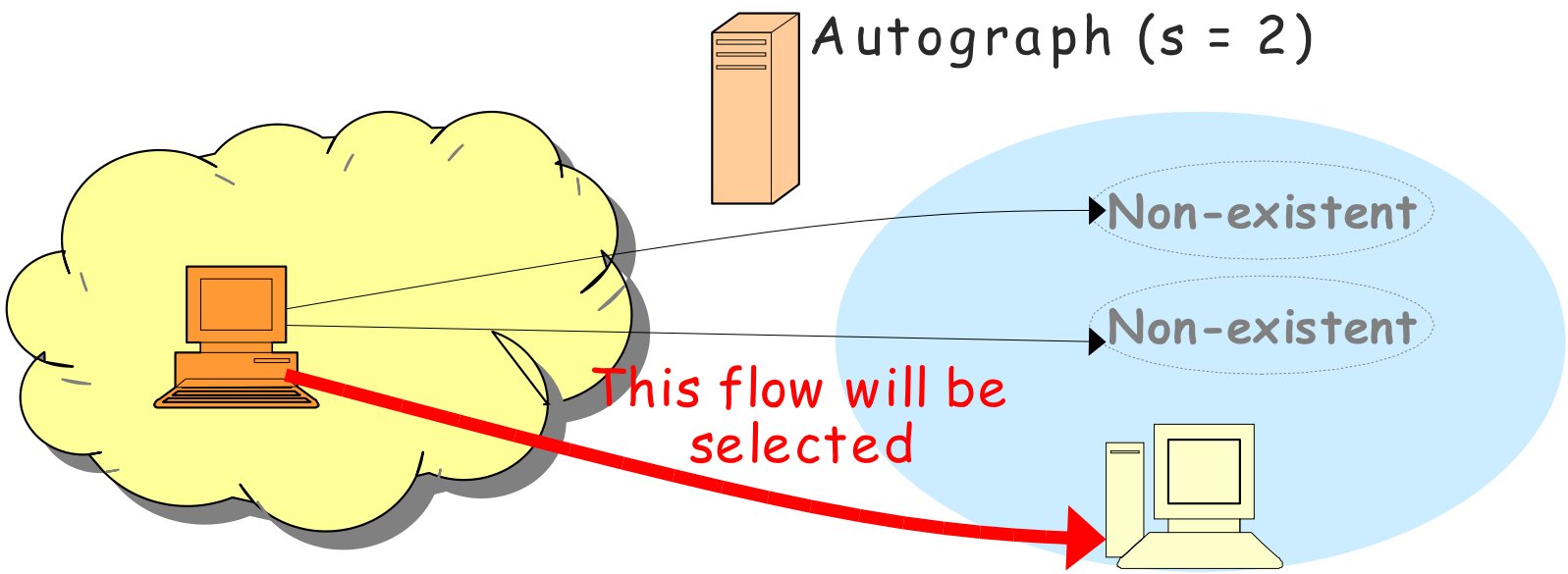
The model



- **Step 1: Select suspicious flows using heuristics**
- **Step 2: Generate signature using content-prevalence analysis**

Suspicious flow selection

- **Idea: Reduce the work by filtering out vast amount of innocuous flows**
- **Heuristic: Flows from scanners are suspicious**
  - Focus on the successful flows from IPs who made unsuccessful connections to more than s destinations for last 24hours
  - Suitable heuristic for TCP worm that scans network



Suspicious flow selection

- **Idea: Reduce the work by filtering out vast amount of innocuous flows**
- **Heuristic: Flows from scanners are suspicious**
  - Focus on the successful flows from IPs who made unsuccessful connections to more than s destinations for last 24hours
  - Suitable heuristic for TCP worm that scans network
- **Suspicious Flow Pool**
  - Holds reassembled, suspicious flows captured during the last time period t
  - Triggers signature generation if there are more than θ flows

Signature generation

- **Idea: Use the most frequent byte sequences across suspicious flows as signatures**
- **All instances of a worm have a common byte pattern specific to the worm**
- **Rationale**
  - Worms propagate by duplicating themselves
  - Worms propagate using vulnerability of a service
- **Problem: How to extract candidate signatures?**
- **Problem: How to select most prevalent signatures?**

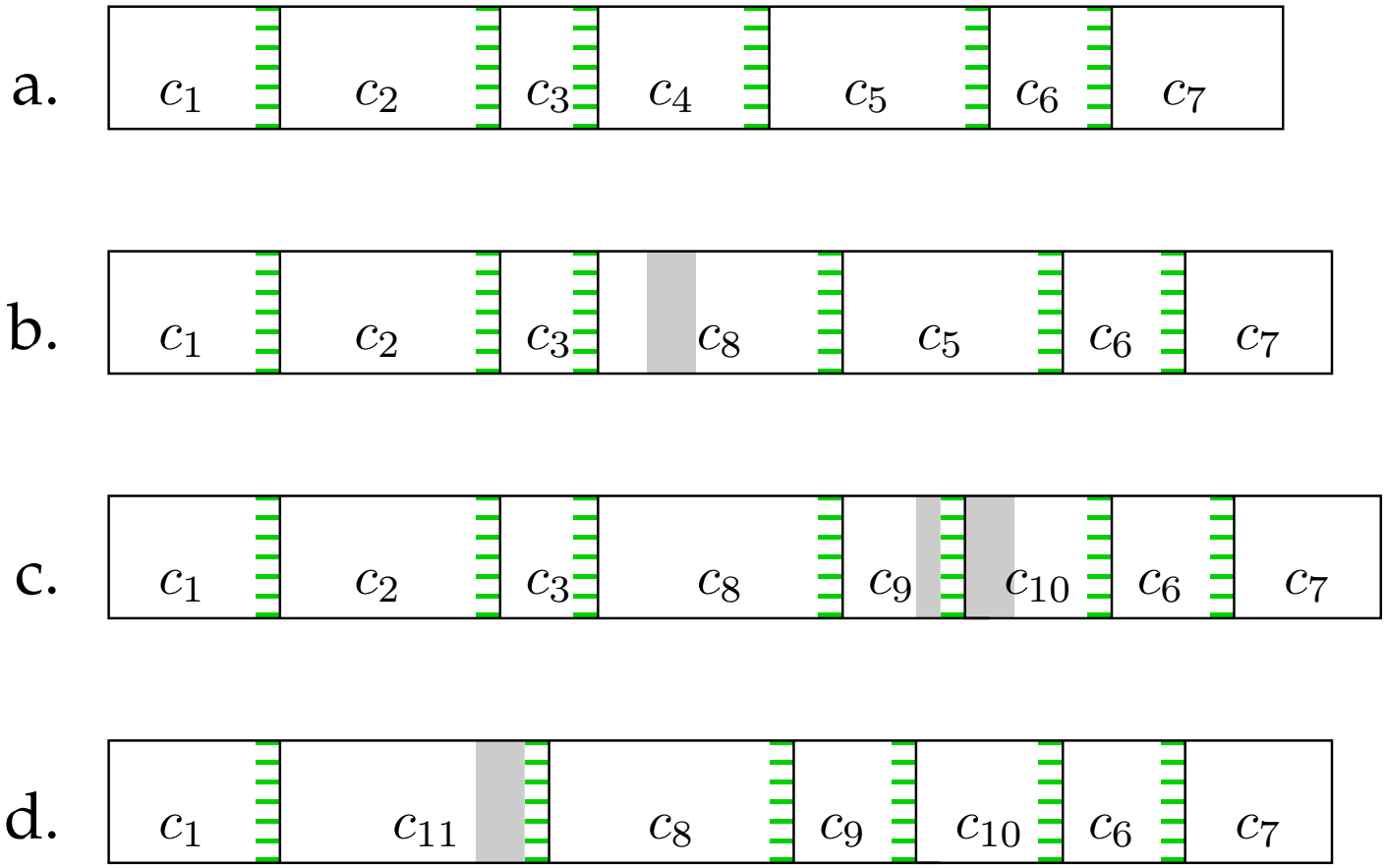
Straw Man

- **Break flow into fixed-size chunks**
- **Keep mapping of** $\mathrm{Hash}(\mathrm{chunk}) \leftrightarrow \mathrm{count}$
- **Sort by count of times seen**
  - Select most seen chunks as worm fingerprints
- **First problem: Alignment**
  - Inserting one byte at start of flow changes all chunks
  - Indexing all overlapping chunks impractical
- **Second problem: Selecting too many signatures**
  - Each signature to check makes monitoring slower
  - May not improve sensitivity
  - May decrease specificity

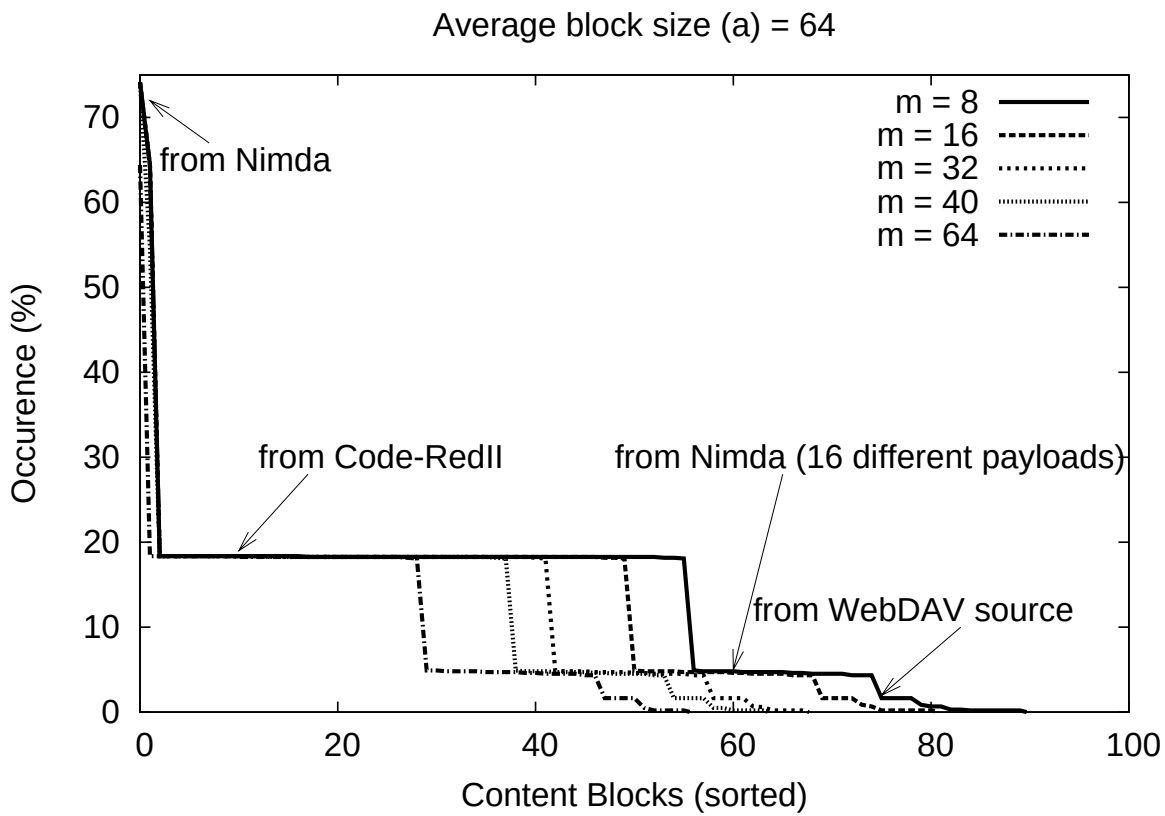
COntent-based Payload Partitioning

- **Base chunks on flow contents, not position**
  - Compute running hash of every overlapping k-byte region
  - If hash mod a is special value B, create chunk boundary
- **Inserting/deleting data only affects that chunk**
- **Problem: Pathological cases**
  - Huge chunk size (e.g., all-zero region)
  - Tiny chunk sizes (e.g., unlucky k-byte region repeated)
- **Impose min/max chunk sizes**
  - Can mess up alignment, but probably OK in practice

Example: slightly different flows



Why prevalence?



- **Worm flows dominate in the suspicious flow pool**
- **Content-blocks from worms are highly ranked**

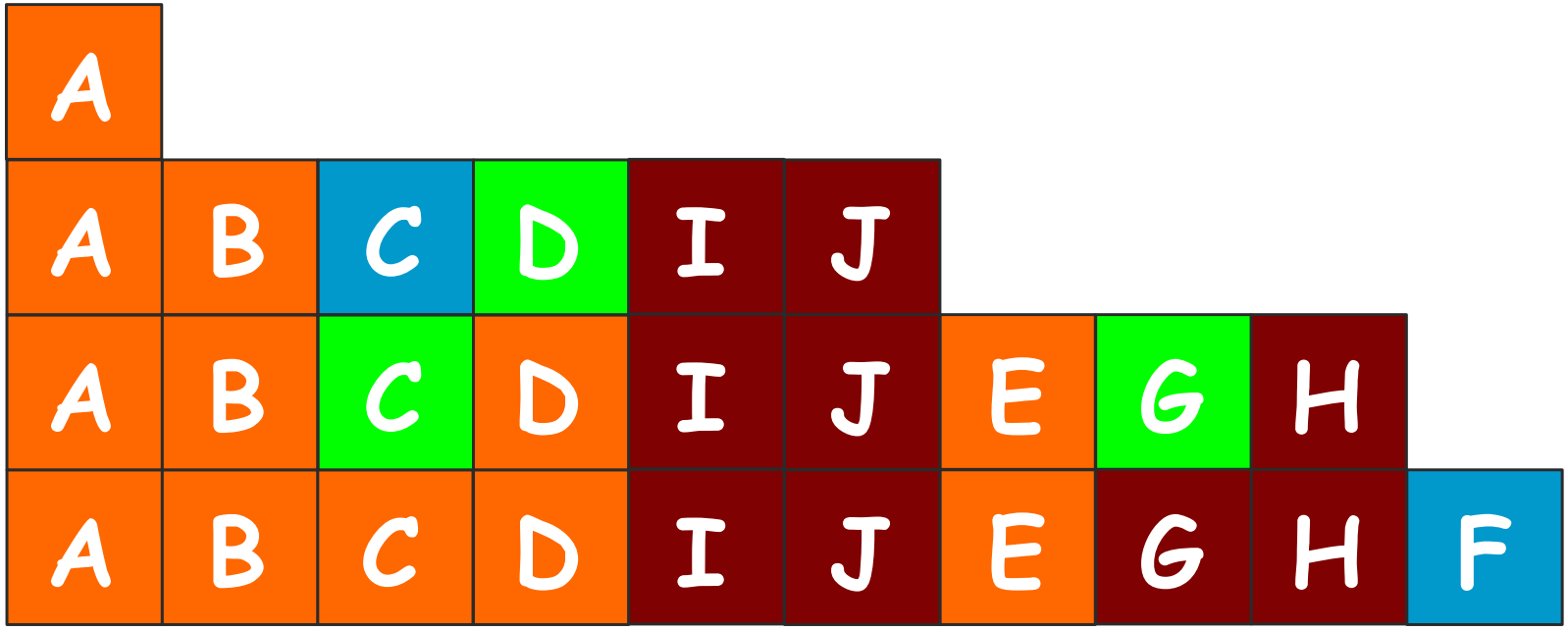
Picking signatures

f0	C	F	
f1	C	D	G
f2	A	B	D
f3	A	C	E
f4	A	B	E
f5	A	B	D
f6	H	I	J
f7	I	H	J
f8	G	I	J

Picking signatures

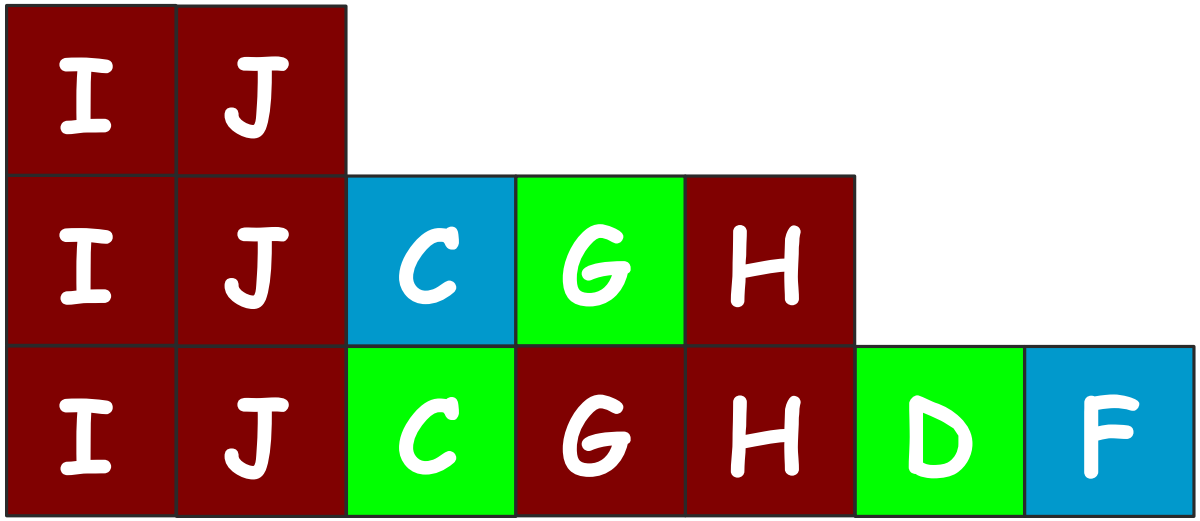
f0	C	F	
f1	C	D	G
f2	A	B	D
f3	A	C	E
f4	A	B	E
f5	A	B	D
f6	H	I	J
f7	I	H	J
f8	G	I	J

Picking signatures



- Sort signatures by prevalence
- Want to cover w fraction of suspicious flows
- Want each signature with $\geq p$ appearances
- Example: Pick A, remove all orange flows

Picking signatures



- **Example: Pick I, remove all maroon flows**
- **If $p = 3$, you are now done**

Results

- **Discuss Figure 6**
- **Looks like** $90\% \leq w \leq 94.8\%$ **very good… Why?**
  - Explain intuitively what happens with low or very high w
- **What are metamorphic and polymorphic worms?**
  - Is there hope for such worms?