STATISTICALLY REGULATING PROGRAM BEHAVIOR VIA MAINSTREAM COMPUTING

Mark Stephenson, Ram Rangan*,
Emmanuel Yashchin, and Eric Van Hensbergen

IBM Research Lab

OUR WORK

THE ELEVATOR PITCH

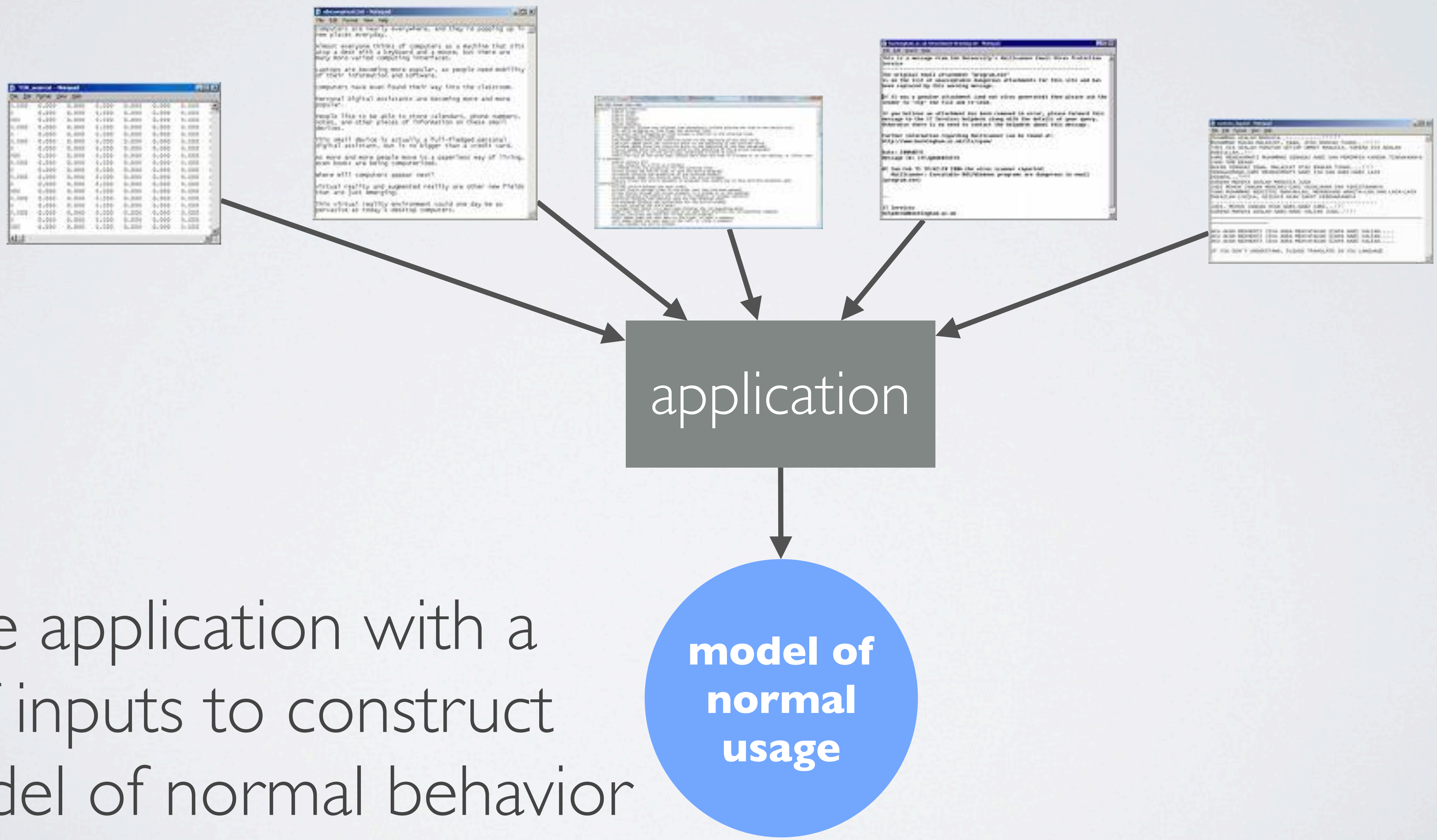
- Build anomaly detection into a deployed application
- Flag the execution of the application if it appears to be abnormal
- Give the user the ability to...
  - adjust the meaning of "abnormal"
  - decide how to proceed when flags are raised

MOTIVATION

ANOMALOUS EXECUTION

- Attacks on vulnerable code
  - Buffer overruns, value overflow and underflow, denial of service, injection attacks, etc.
- Soft errors

MOTIVATION
ANOMALY DETECTION



Profile application with a set of inputs to construct a model of normal behavior

MOTIVATION
ANOMALY DETECTION

- Models for anomaly detection are trained on a set of inputs (called the *training set*)
  - Generally, training with more inputs reduces false positives
  - ...but, increases the number of false negatives of the model
- **Current systems don't give the user a method for adjusting the tradeoff between the two "falses"**

MOTIVATION
ANOMALY DETECTION

- Models for anomaly detection are trained on a set of inputs (called the *training set*)
  - Generally, training with more inputs reduces false positives
  - ...but, increases the number of false negatives of the model
- **Current systems don't give the user a method for adjusting the tradeoff between the two "falses"**

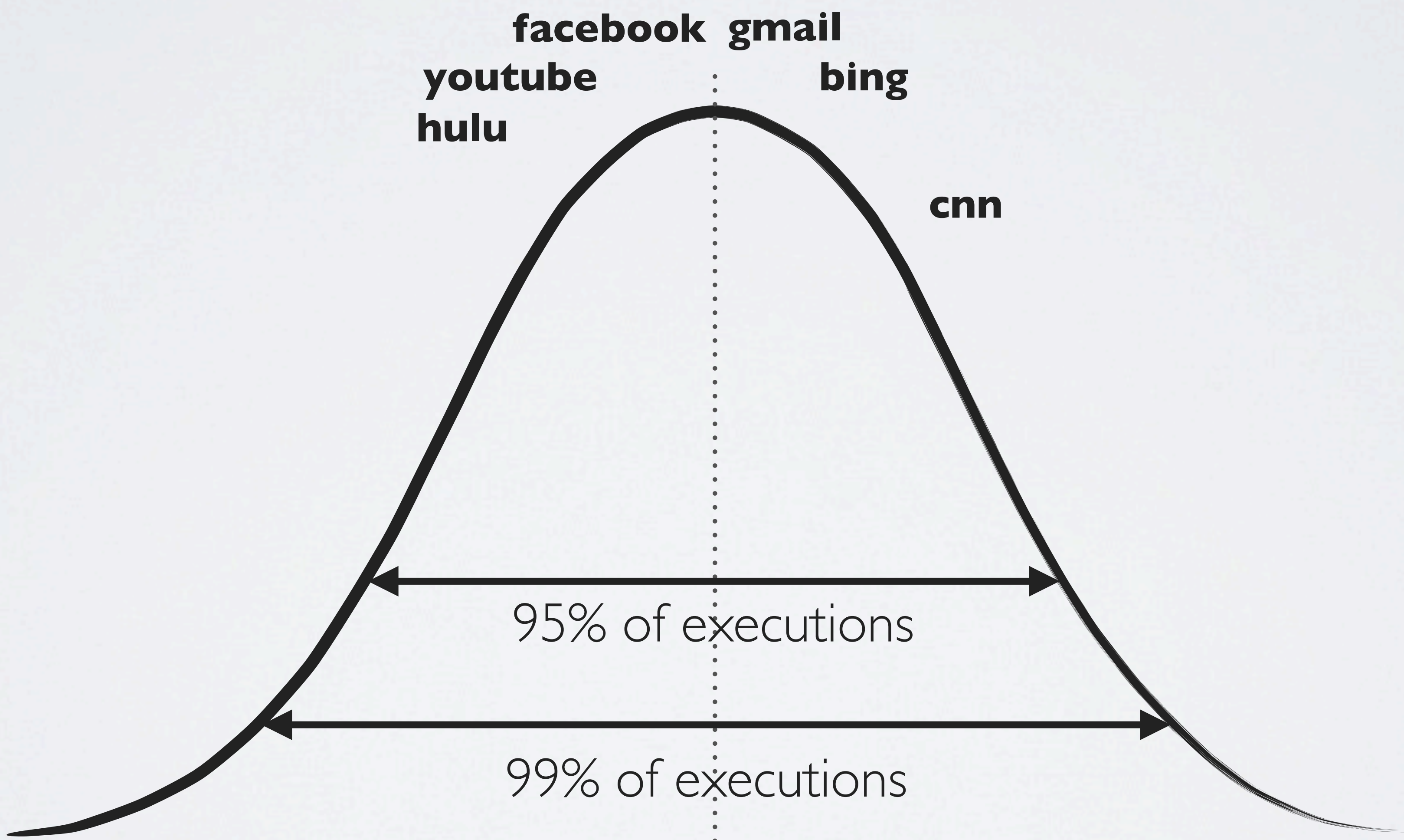
MOTIVATION

ANOMALY DETECTION

- Models for anomaly detection are trained on a set of inputs (called the *training set*)
  - Generally, training with more inputs reduces false positives
  - ...but, increases the number of false negatives of the model
- **Current systems don't give the user a method for adjusting the tradeoff between the two "falses"**

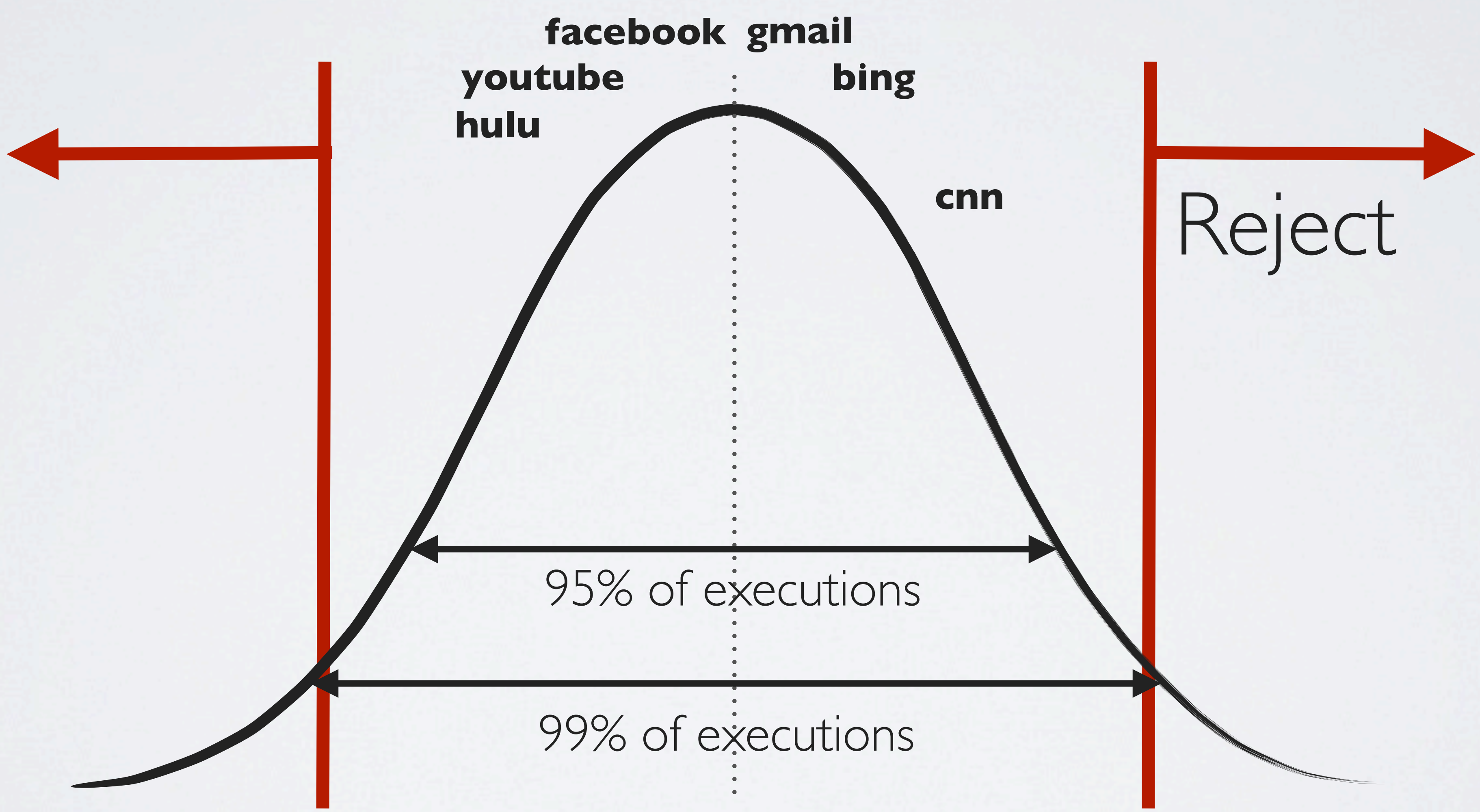
MAINSTREAM COMPUTING

CONCEPTUAL FIGURE



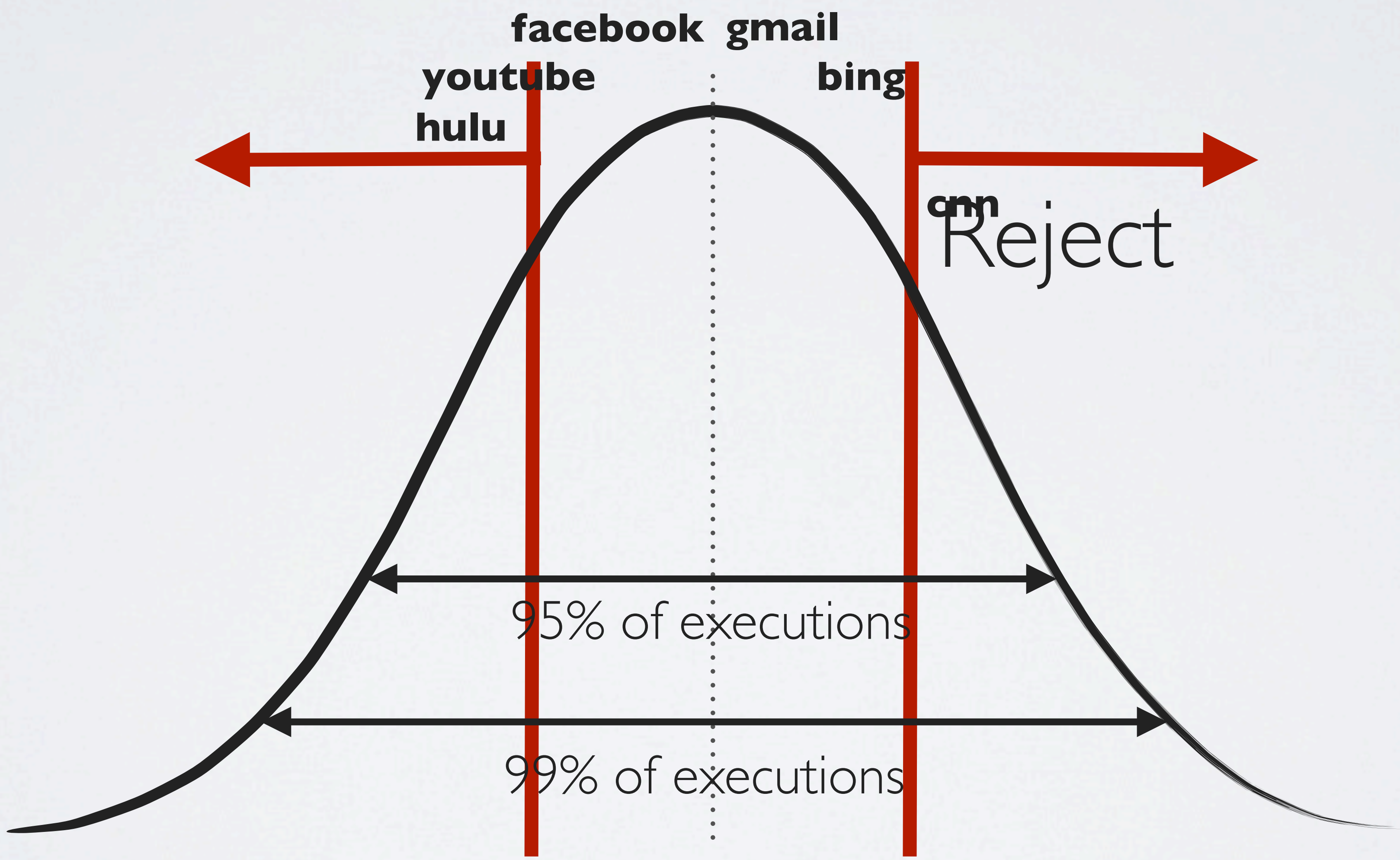
MAINSTREAM COMPUTING

CONCEPTUAL FIGURE



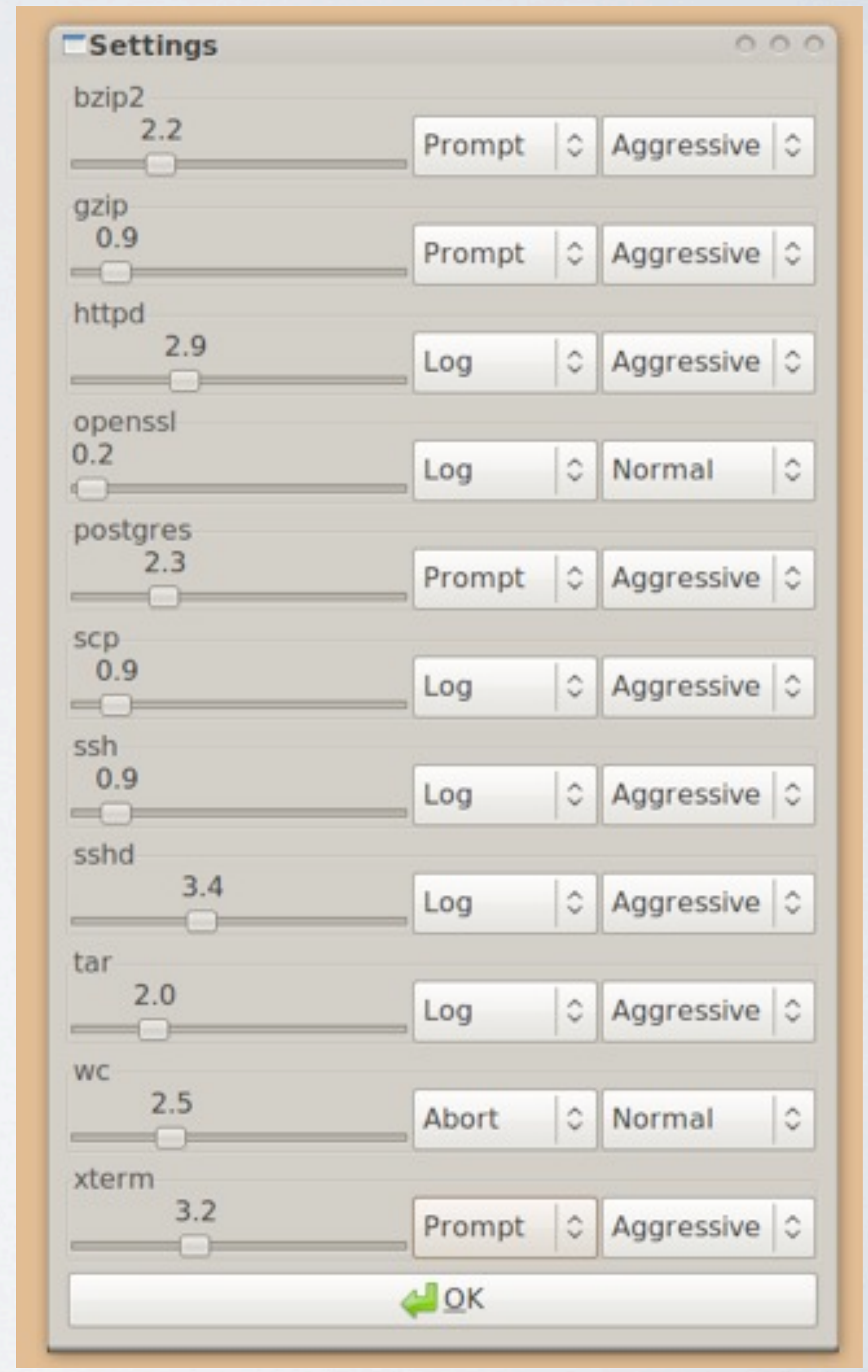
MAINSTREAM COMPUTING

CONCEPTUAL FIGURE



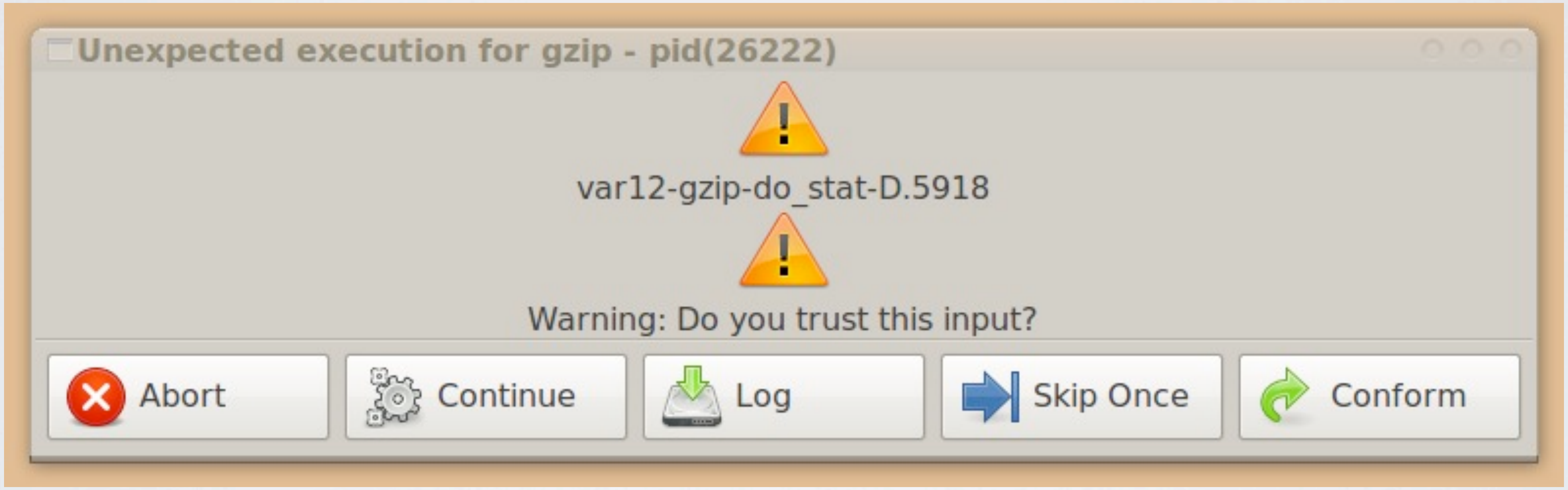
OUR SYSTEM
MAINSTREAM COMPUTING

- Allow a user to say, "Ensure that this execution *conforms* with 99% of the usage patterns for the application"
- System constructs a model that is *statistically* guaranteed to raise a flag at most 1% of the times the application is invoked
- Provide a single knob for each application
- Allow the user to select what action is taken when a flag is raised



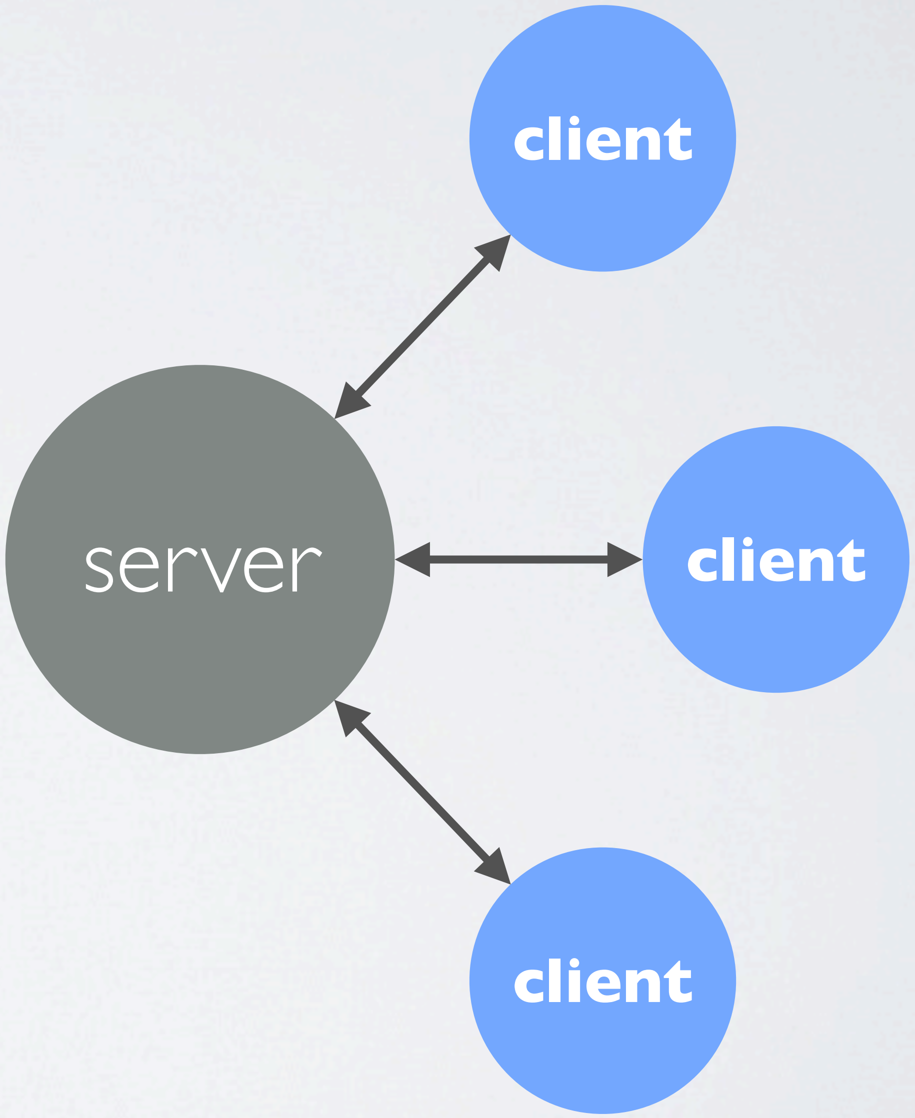
RECOURSE

- What should we do when the execution looks abnormal?



HIGH LEVEL VIEW

- Collaborative approach
- A centralized server collects *runtime profiles* from clients
- Centralized server uses these runtime profiles to generate *constraint sets* for applications

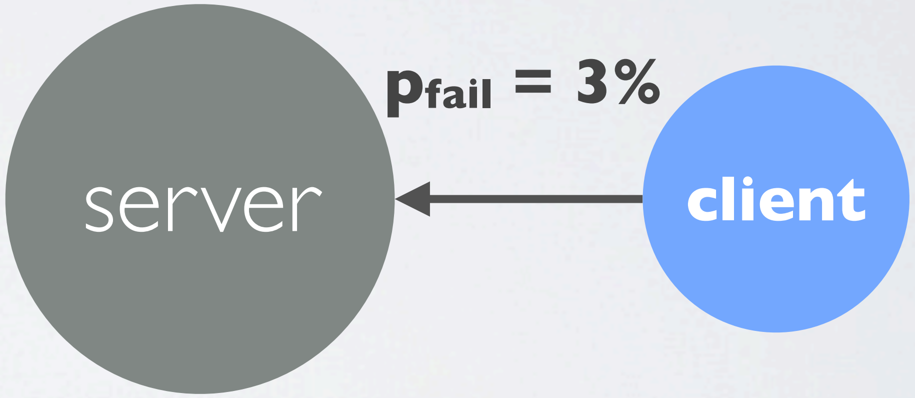


CLIENT OPERATION

- Clients have two main tasks:

  1. Ensure that server provided constraints are not violated

  2. Continually sample aspects of execution for *this* invocation

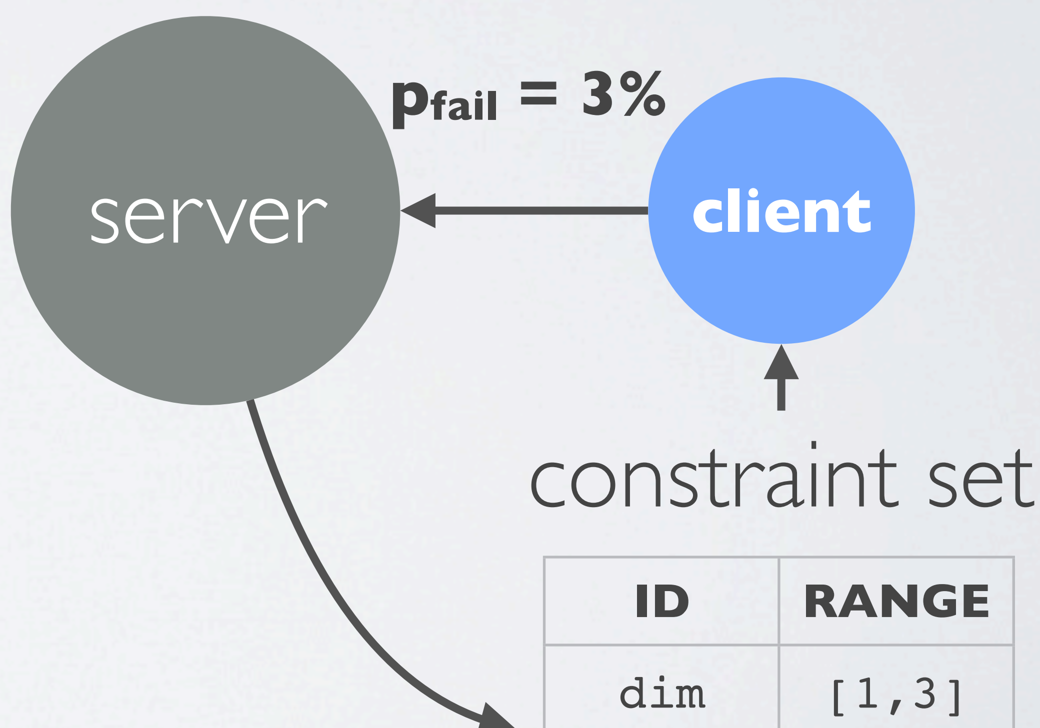


CLIENT OPERATION

- Clients have two main tasks:

  1. Ensure that server provided constraints are not violated

  2. Continually sample aspects of execution for *this* invocation



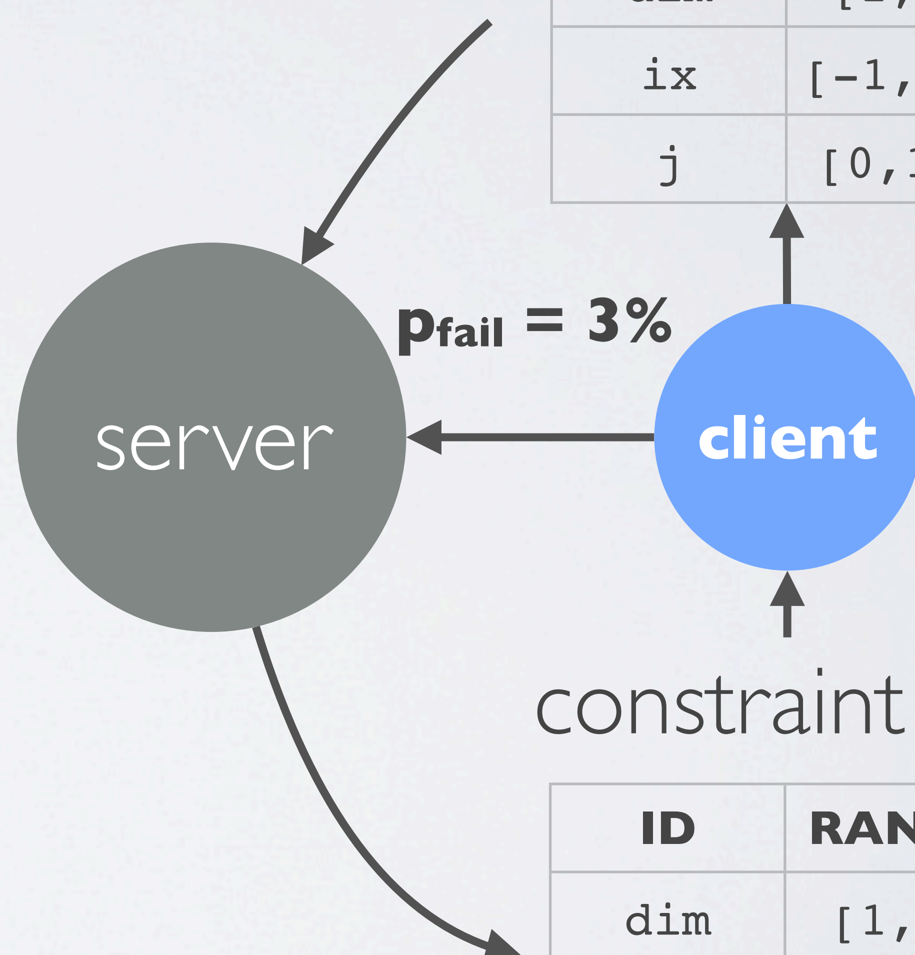
ID	RANGE
dim	[1,3]
ix	[-1,99]
j	[0,99]

CLIENT OPERATION

- Clients have two main tasks:
  1. Ensure that server provided constraints are not violated
  2. Continually sample aspects of execution for *this* invocation

runtime profile

ID	RANGE
dim	[2,3]
ix	[-1,52]
j	[0,19]



constraint set

ID	RANGE
dim	[1,3]
ix	[-1,99]
j	[0,99]

SERVER OPERATION

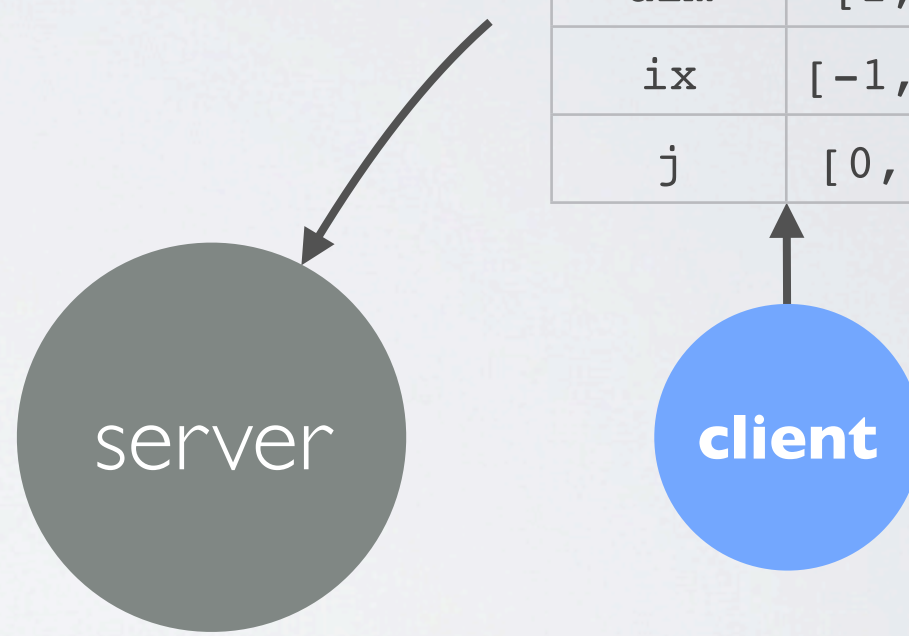
- Server aggregates runtime profiles, from the clients

- **Creates constraint sets with *statistical bounds* on failure rates**

- **Can probabilistically tolerate runtime profiles from *rogue* users**

runtime profile

ID	RANGE
dim	[2,8]
ix	[-1,52]
j	[0,19]



SERVER OPERATION

runtime profiles

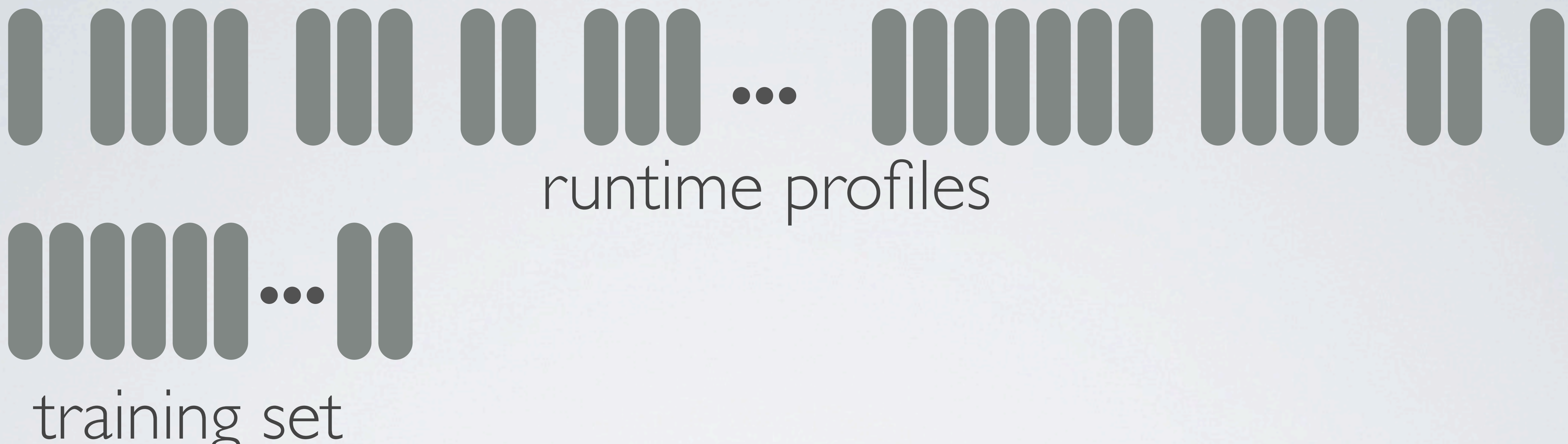
- Server aggregates runtime profiles, from the clients

SERVER OPERATION

runtime profiles

- Separate runtime profiles into a *training set* and a *validation set*
- These sets are disjoint

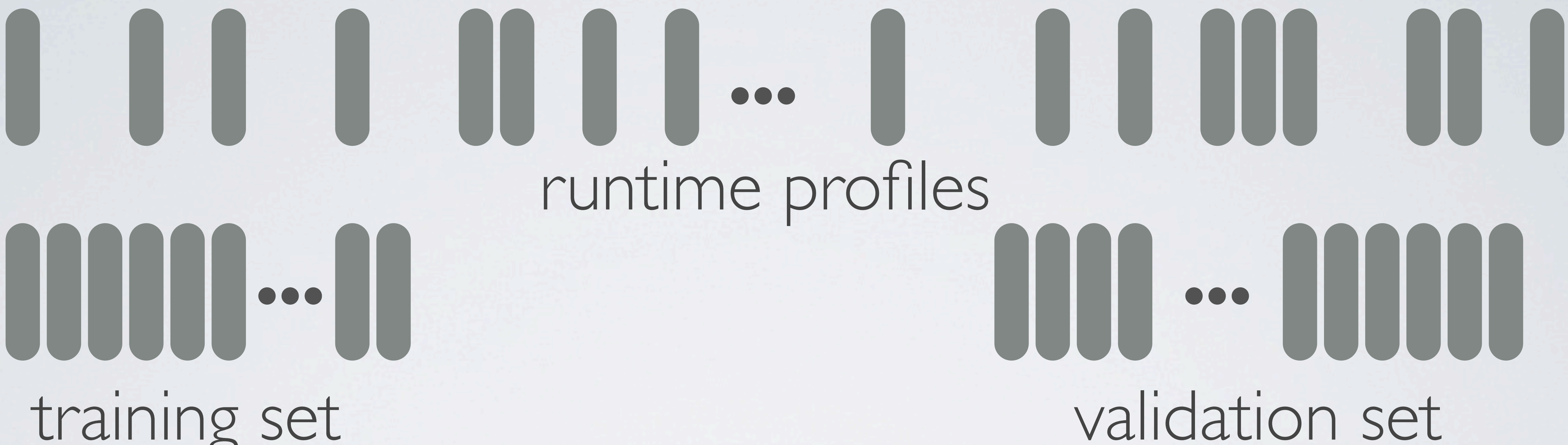
SERVER OPERATION



- Separate runtime profiles into a *training set* and a *validation set*

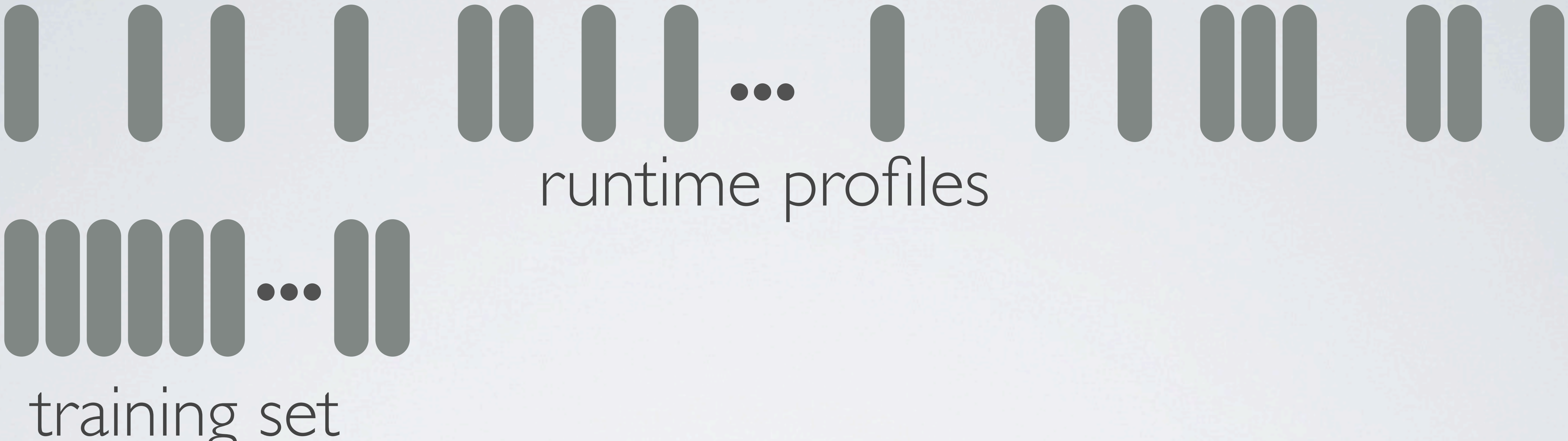
- These sets are disjoint

SERVER OPERATION



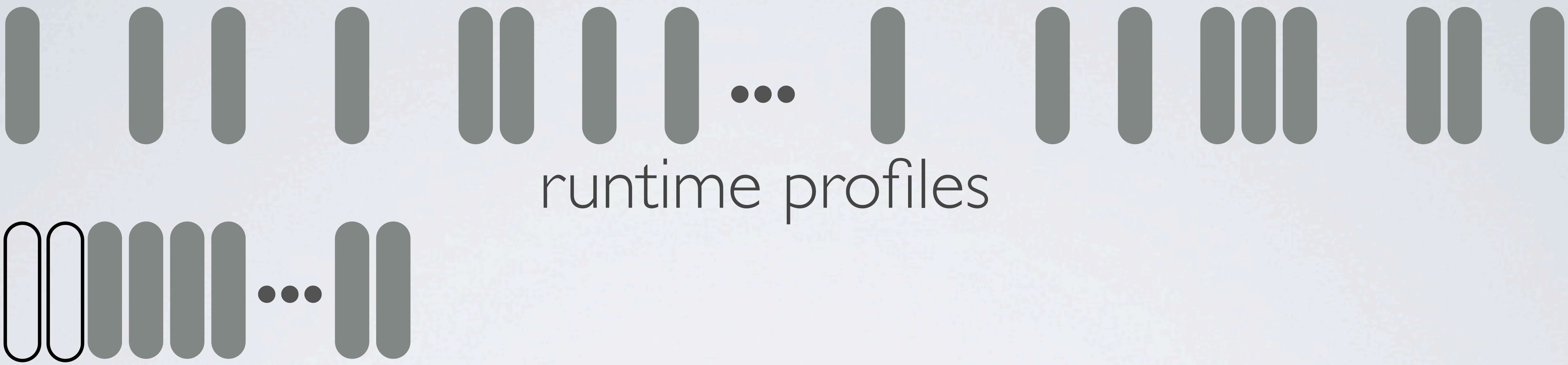
- Separate runtime profiles into a *training set* and a *validation set*
- These sets are disjoint

SERVER OPERATION



- Create a model of nominal behavior using runtime profiles in the training set

SERVER OPERATION



runtime profiles

training set

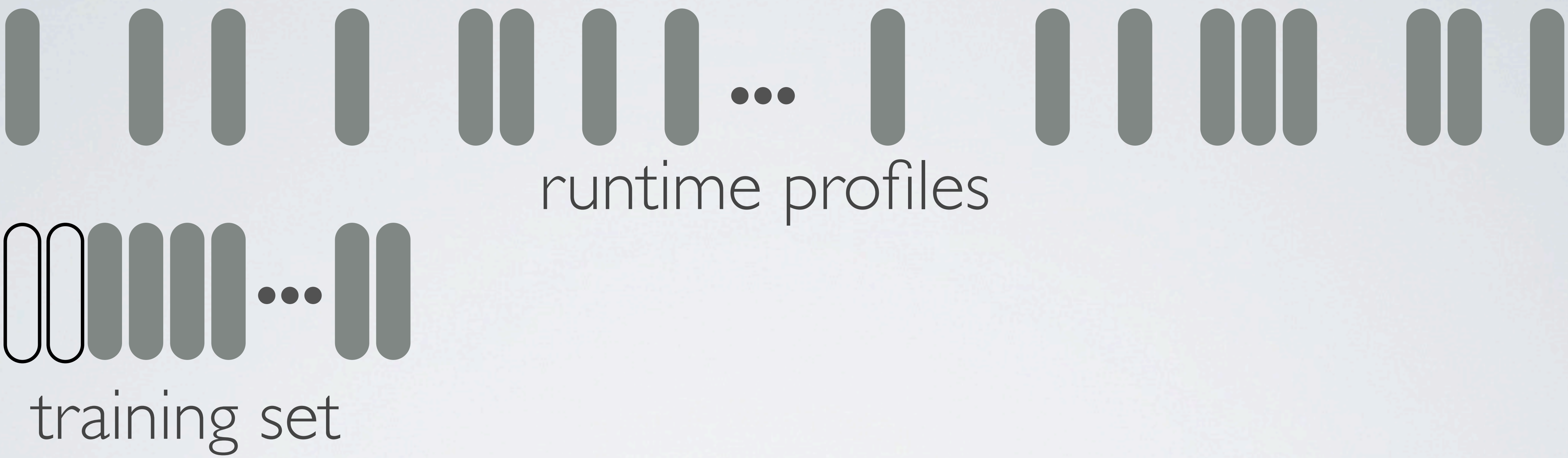
ID	RANGE
dim	[2,3]
ix	[-1,10]
j	[2,11]

∪

ID	RANGE
dim	[1,2]
ix	[0,99]
j	[1,99]

- Create a model of nominal behavior using runtime profiles in the training set

SERVER OPERATION



ID	RANGE
dim	[2,3]
ix	[-1,10]
j	[2,11]

$\cup$

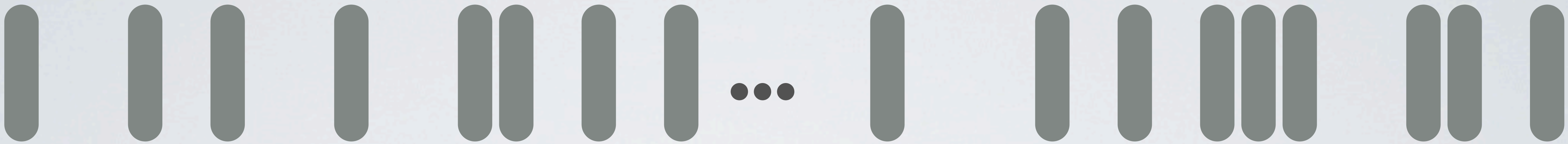
ID	RANGE
dim	[1,2]
ix	[0,99]
j	[1,99]

$=$

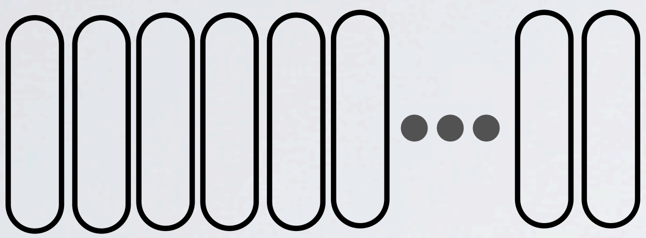
ID	RANGE
dim	[1,3]
ix	[-1,99]
j	[1,99]

- Create a model of nominal behavior using runtime profiles in the training set

SERVER OPERATION



runtime profiles



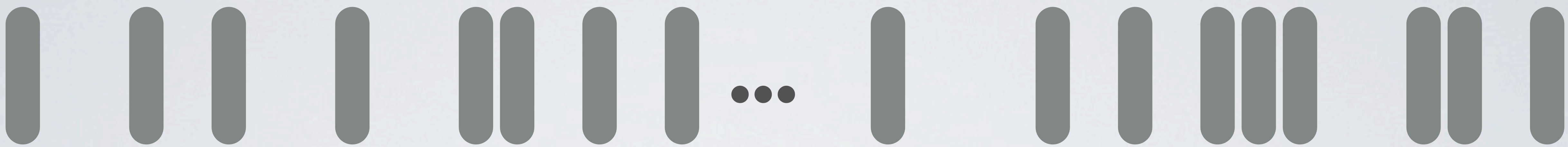
training set

validation set

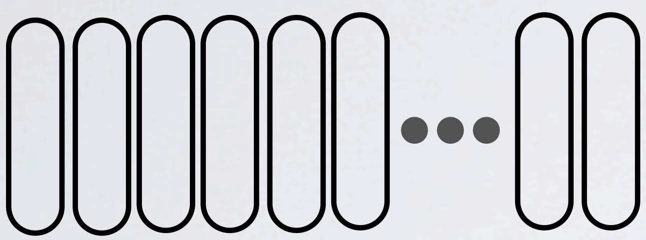
ID	RANGE
`dim`	`[1,12]`
`ix`	`[-2,99]`
`j`	`[0,99]`

constraint set

SERVER OPERATION



runtime profiles



training set

validation set

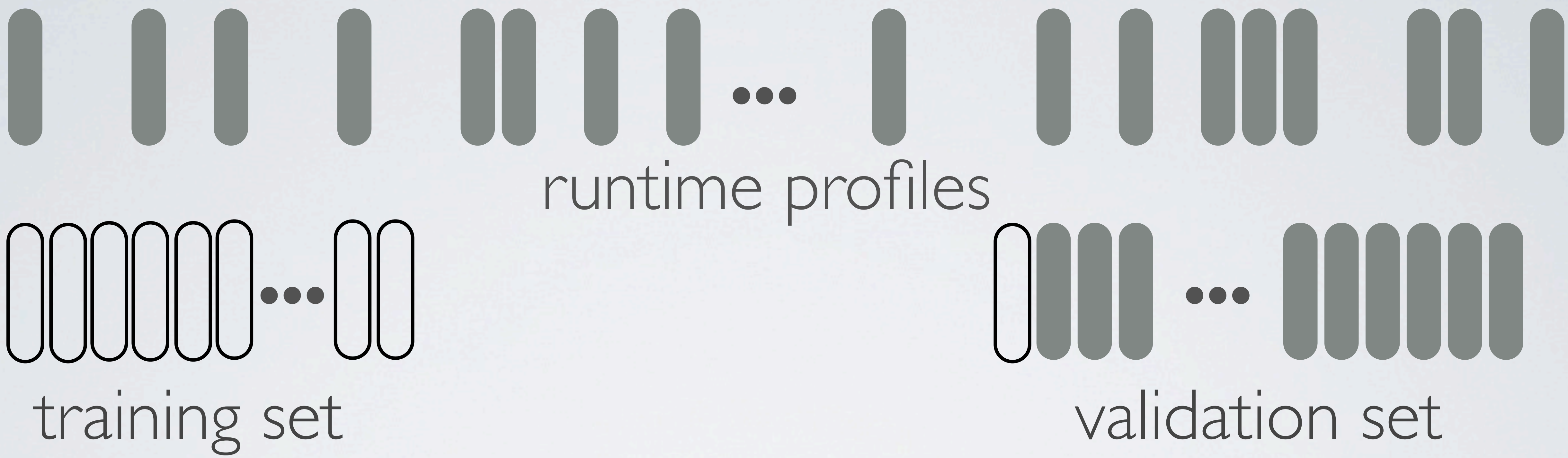
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

ID	RANGE
dim	[1,12]
ix	[1,50]
j	[3,91]

constraint set

SERVER OPERATION



ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

ID	RANGE
dim	[1,12] ✔
ix	[1,50] ✔
j	[3,91] ✔

constraint set

SERVER OPERATION

runtime profiles

training set

validation set

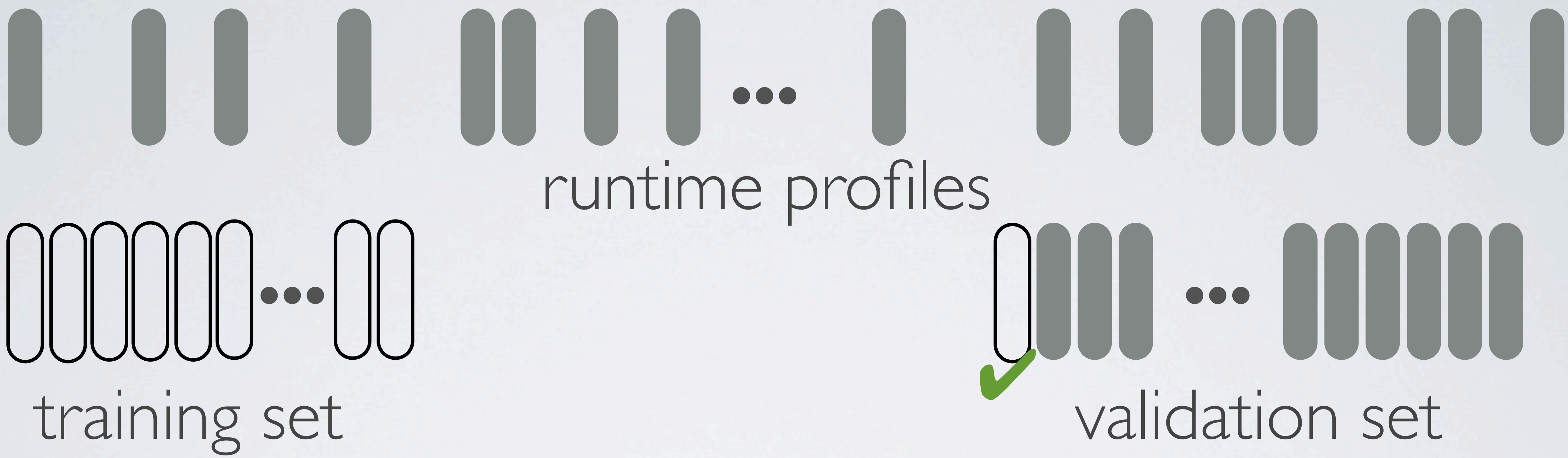
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

ID	RANGE
dim	[1,12]
ix	[1,50]
j	[3,91]

constraint set

SERVER OPERATION

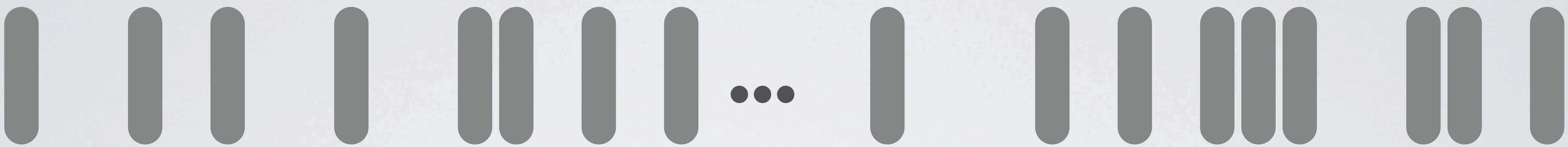


ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

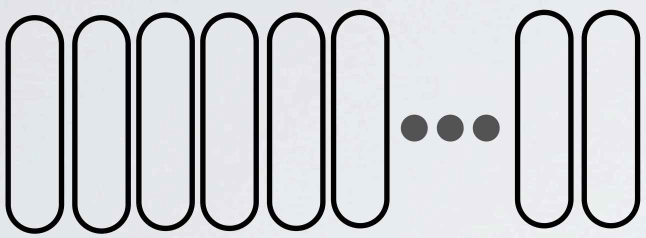
$\supseteq$

constraint set

SERVER OPERATION



runtime profiles



training set

validation set

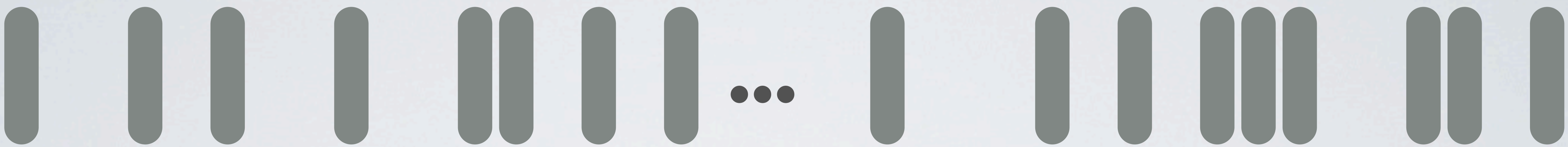
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

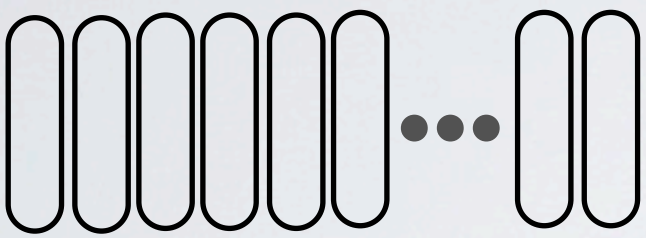
ID	RANGE
dim	[0,11]
ix	[1,50]
j	[3,91]

constraint set

SERVER OPERATION



runtime profiles



training set

validation set

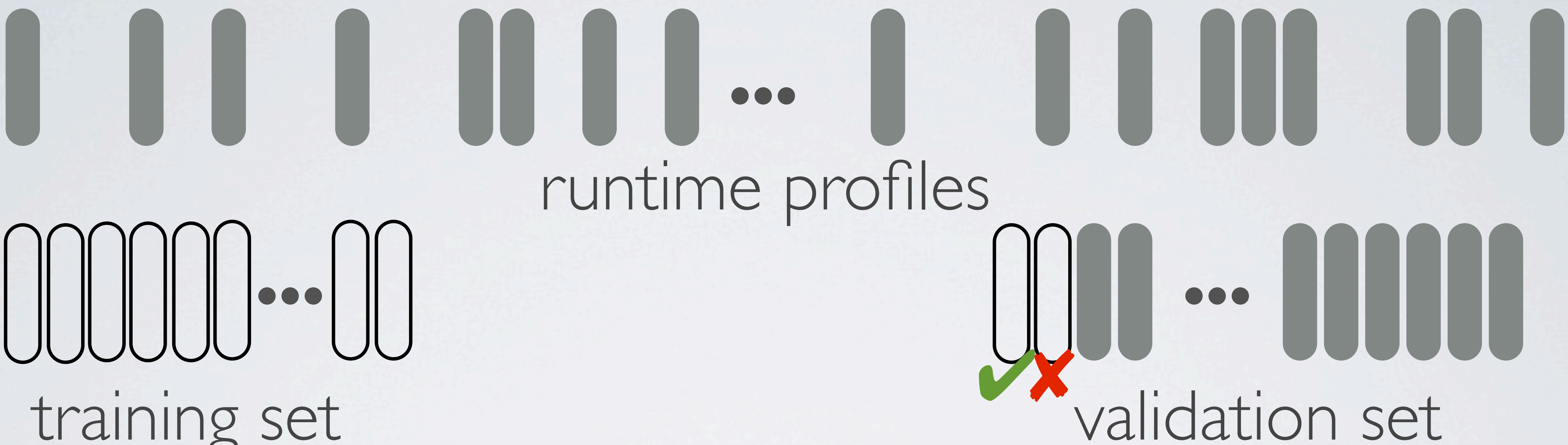
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

ID	RANGE
dim	[0,11]
ix	[1,50]
j	[3,91]

constraint set

SERVER OPERATION



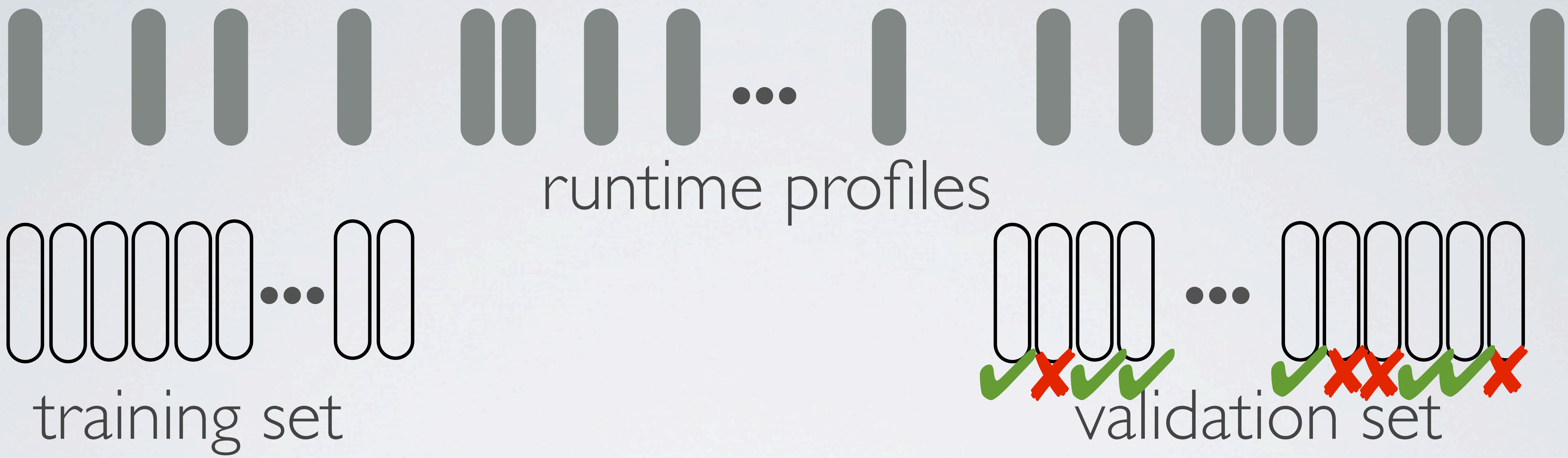
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

$\supseteq$

ID	RANGE
dim	[0,11]
ix	[1,50]
j	[3,91]

constraint set

SERVER OPERATION

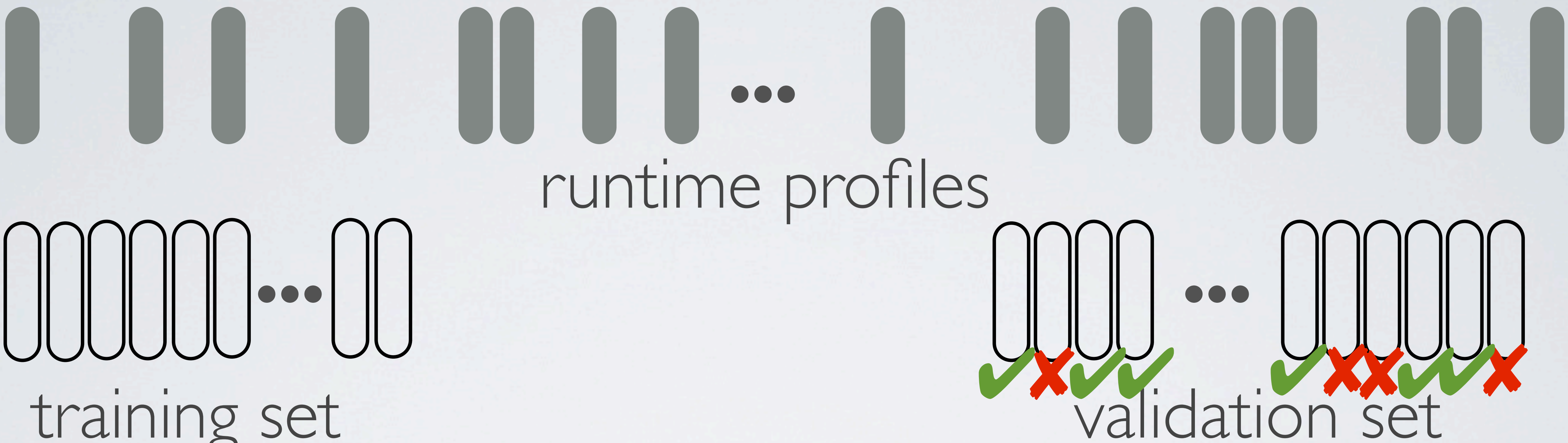


ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

constraint set

$$\hat{P}_{fail} = \frac{N_{failed}}{N_{validate}}$$

SERVER OPERATION



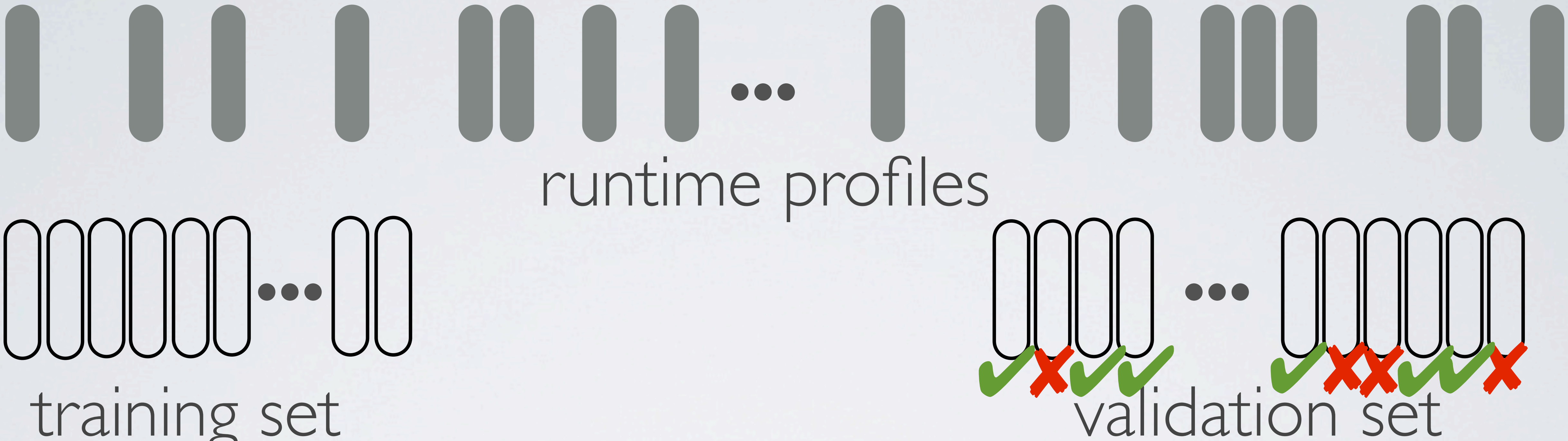
ID	RANGE
`dim`	`[1,12]`
`ix`	`[-2,99]`
`j`	`[0,99]`

constraint set

$$\hat{P}_{fail} = \frac{N_{failed}}{N_{validate}}$$

- This is only an *estimate*, the accuracy of which depends on $N_{validate}$

SERVER OPERATION



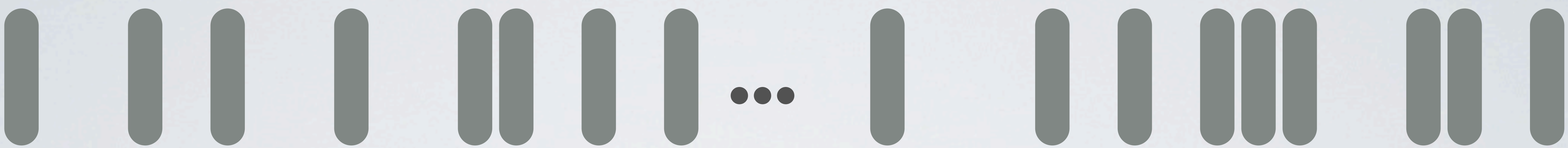
ID	RANGE
`dim`	`[1,12]`
`ix`	`[-2,99]`
`j`	`[0,99]`

constraint set

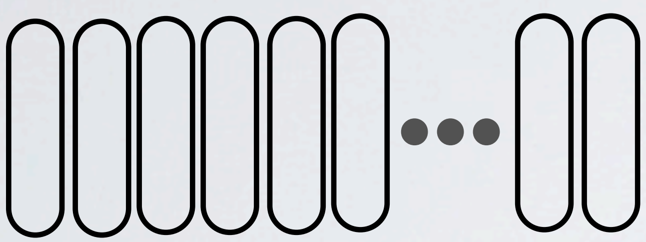
$$\hat{P}_{fail} = \frac{N_{failed}}{N_{validate}} \pm \epsilon$$

- We can find a statistical upper bound for the failure rate by using the well-known solution to the *polling problem*

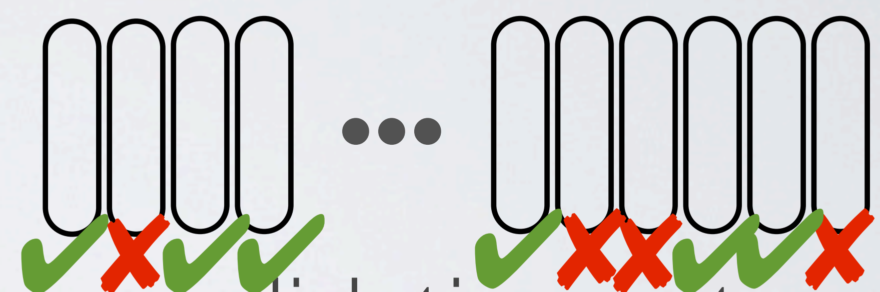
SERVER OPERATION



runtime profiles



training set



validation set

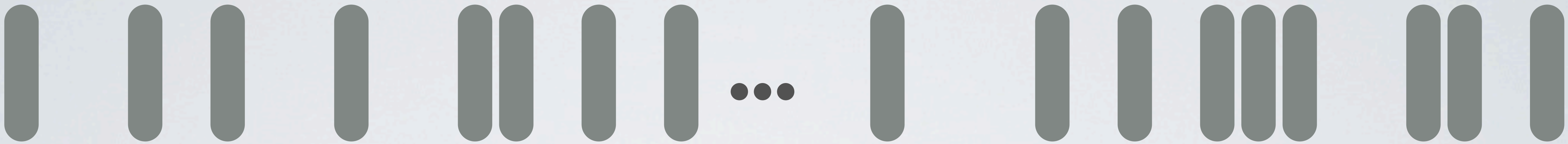
ID	RANGE
dim	[1,12]
ix	[-2,99]
j	[0,99]

constraint set

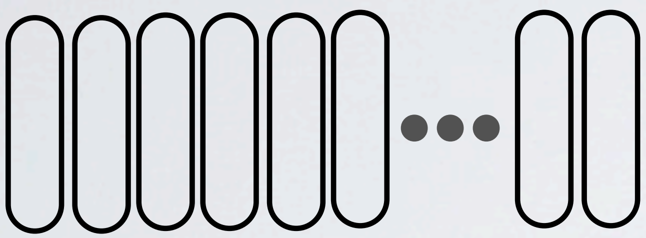
SET	N_{train}	$\hat{P}_{fail}$
	100	35.42
	200	22.98
⋮	⋮	⋮
	3900	0.10
	4000	

$$\hat{P}_{fail} = \frac{N_{failed}}{N_{validate}} \pm \epsilon$$

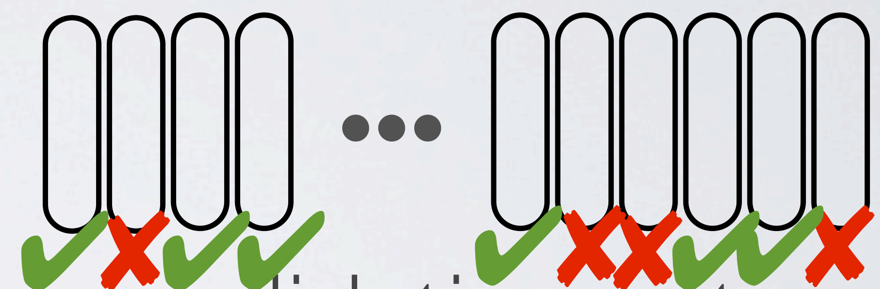
SERVER OPERATION



runtime profiles



training set



validation set

SET	N_{train}	$\hat{P}_{fail}$
	100	35.42
	200	22.98
⋮	⋮	⋮
	3900	0.10
	4000	

$$\hat{P}_{fail} = \frac{N_{failed}}{N_{validate}} \pm \epsilon$$

constraint set

PROTOTYPE IMPLEMENTATION

- Augmented GCC (version 4.2) with a *mainstream computing* pass
  - Pass inserts calls to a runtime library that simultaneously sample execution and ensure constraints aren't violated
  - Object file constructor modified to initialize execution constraints
- Communication with server implemented as a daemon
  - When user changes tolerances, daemon fetches latest constraint set from the server
  - Similarly, it periodically pushes client runtime profiles back to the server

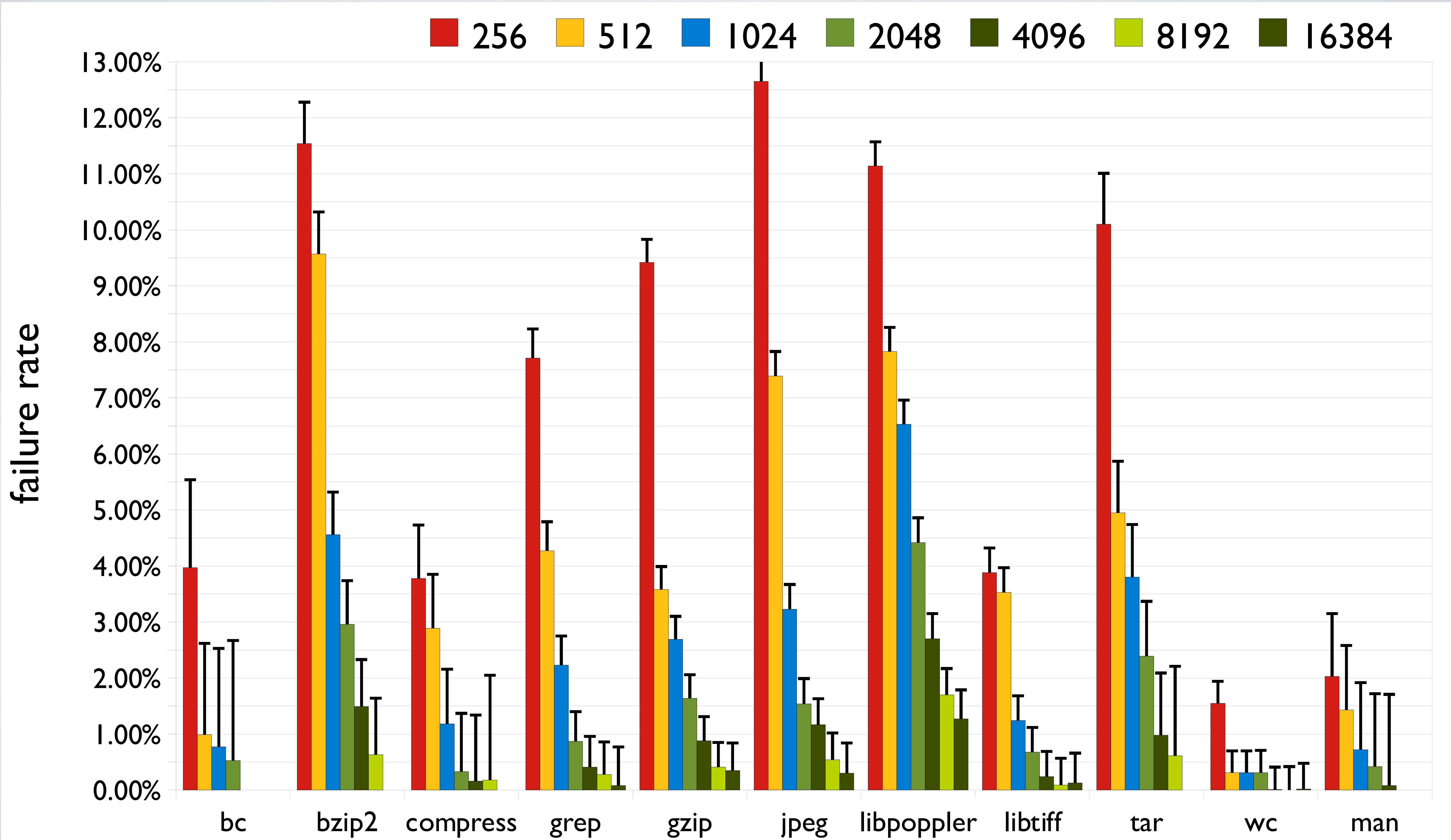
RESULTS

- Perform the following experiments
  - **False positive study**
  - **Detecting exploits**

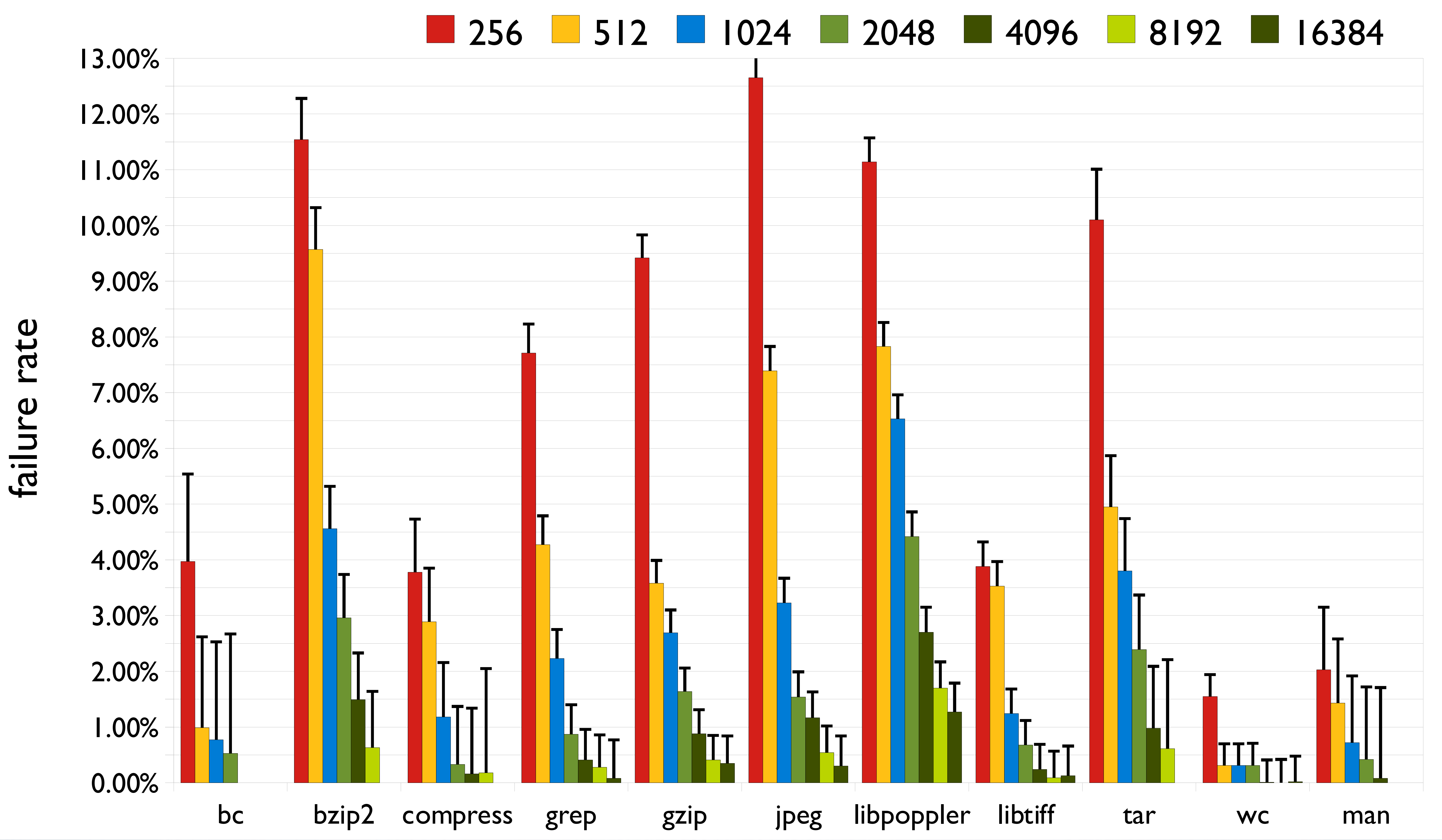
  tradeoff

  - Detecting soft errors, failure oblivious execution, runtime overhead
- We simulate a user community

$\hat{P}_{fail}$: FALSE POSITIVES



$\hat{P}_{fail}$: FALSE POSITIVES

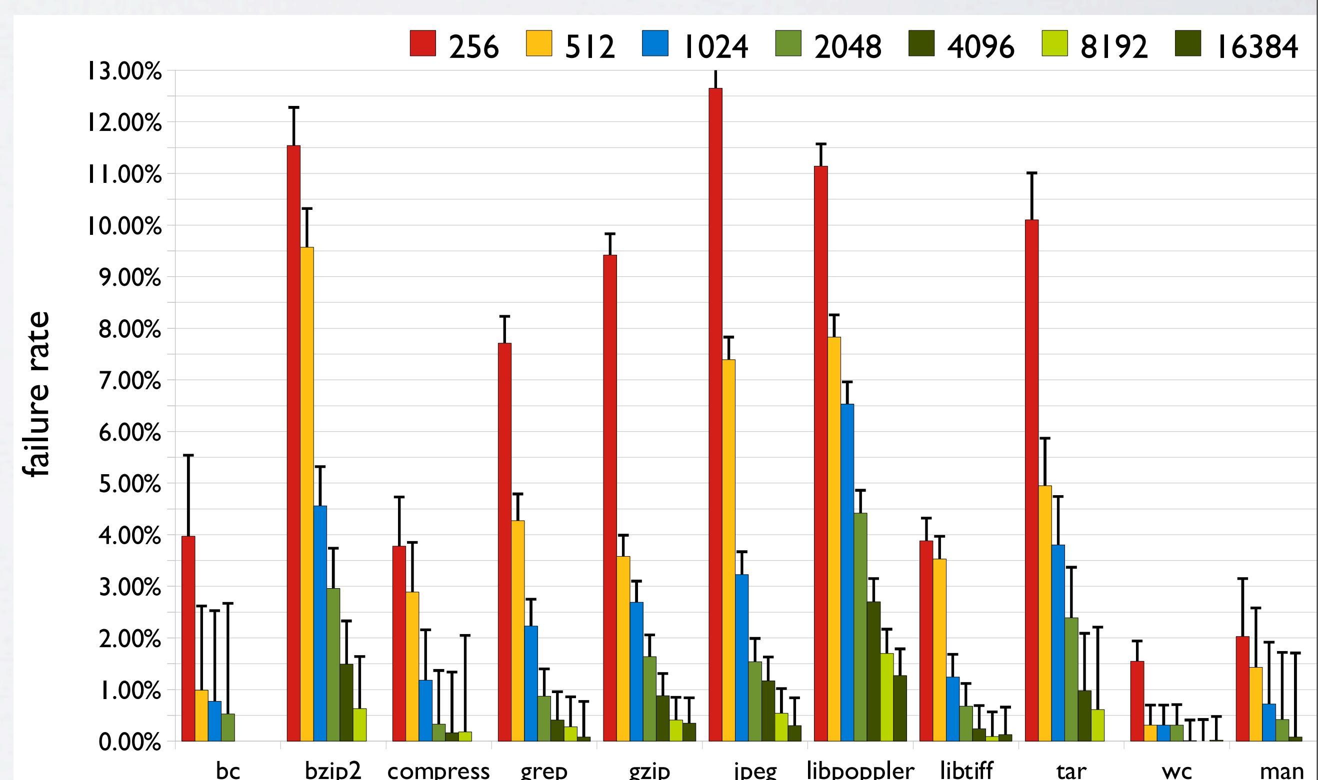


$\hat{P}_{fail}$: FALSE POSITIVES

- Future work to reduce failure rates:

1. Only instrument likely indicator variables

2. Use smoke detector model



DETECTING EXPLOITS

app	exp	256	512	1024	2048	4096	8192	16384
bc	BV	100%	100%	100%	100%	NA	NA	NA
compress	BV	0%	0%	0%	0%	0%	0%	NA
grep	DOS	100%	100%	100%	100%	100%	100%	100%
gzip	BV	100%	50%	40%	30%	10%	0%	0%
libpoppler	UPF	100%	100%	100%	100%	100%	100%	100%
libtiff	OVF	100%	100%	100%	100%	100%	100%	100%
man	BV	100%	100%	100%	100%	100%	NA	NA

RELATED WORK

- **Forrest et al.** *The Evolution of System-call Monitoring.* ACSAC 2008
- **Perkins et al.** *Automatically Patching Errors in Deployed Software*, SOSP 2009
- **Demsky et al.** *Inference and Enforcement of Data Structure Consistency Specifications*, ISSTA '06
- Key differences:
  - Allow user to tradeoff false positives for false negatives
  - Demonstrate ability to thwart several types of attacks and soft errors
  - Analytically show that we can tolerate rogue users in the community

CONCLUSIONS

- We need systems that can identify exploits in deployed code
- Allow users to specify failure rates they are willing to tolerate
- Mainstream computing can identify unanticipated, and potentially malicious execution
  - Buffer overruns, integer overflow, injection attacks, and DOS
- We show that it can even be used to identify soft errors

FUTURE WORK

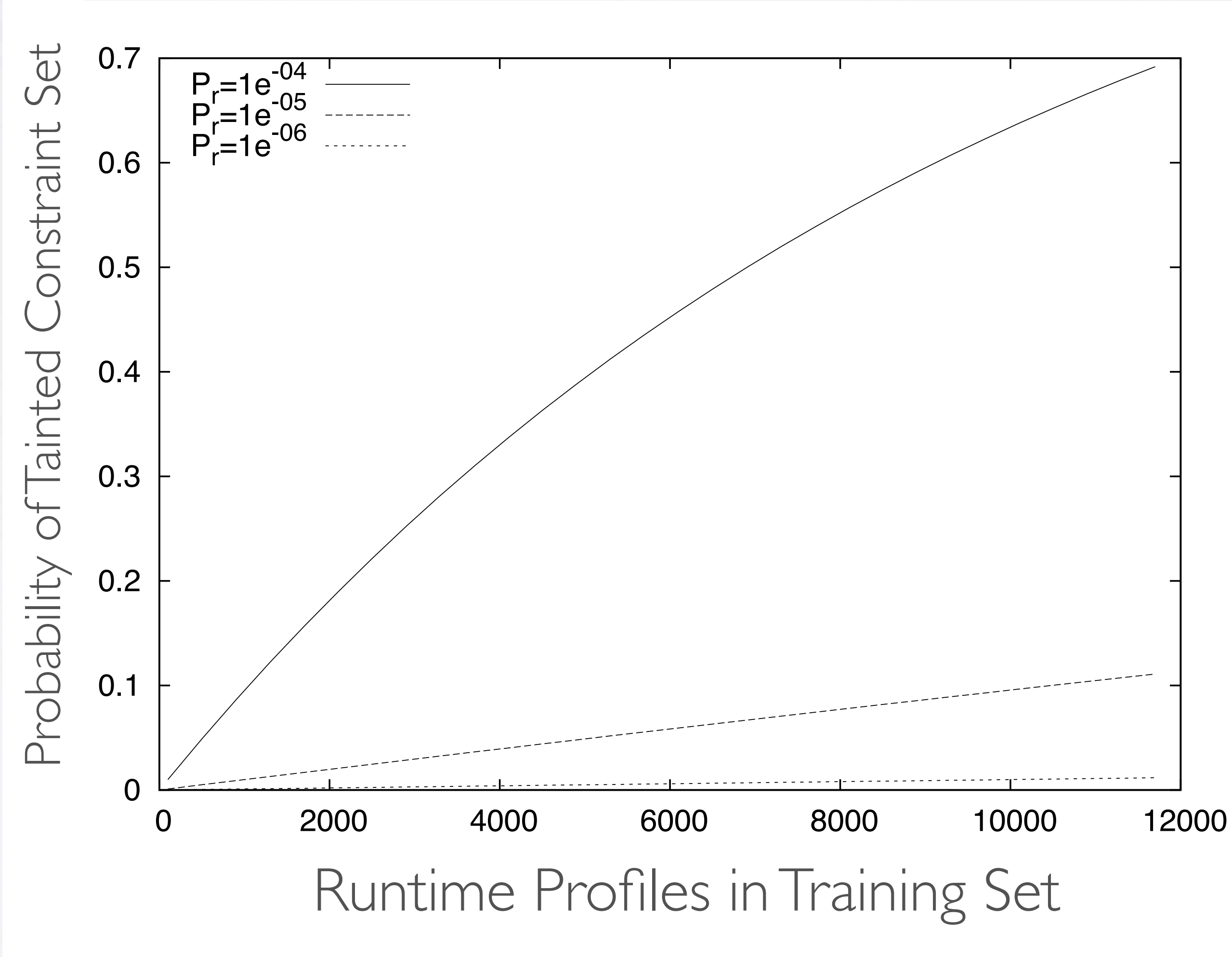
- Only instrument the likely indicator variables
- Deploy in the "real world"
- Consider server workloads
- Improve the performance of the runtime
- Consider privacy concerns

FILTERING VOLATILE VARIABLES

- Our prototype samples nearly all variables
- Some variables (e.g., timing-based variables) are hard or impossible to constrain with our baseline strategy
- We use a machine-learning strategy for filtering them out of the constraint set
- Future work will obviate the need for this strategy

TOLERATING ROGUE USERS

- Community may have malicious users
- Our constraint set creation approach attempts to limit the number of runtime profiles in the training set
- The fewer the runtime profiles in the training set, the less likely it is that the resultant constraint set will be tainted by rogue runtime profiles



ASPECTS OF EXECUTION

VALUE BASED

- *Sample* and *constrain* the values of program "variables"
- Variables: application-level and many IR temporaries
- What we sample and constrain:
  - Data-range: e.g., [32, 36]
  - Constant bits: e.g., [00100TTT]
  - Population count range: e.g., [1, 2]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8]	[0000TTTT]	[1,1]	2 [00000010]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT]	[1,1]	2 [00000010]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1]	2 [00000010]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8]	[0000TTTT]	[1,1]	8 [00001000]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT]	[1,1]	8 [00001000]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1]	8 [00001000]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	8 [00001000]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	8 [00001000]
[1,8]	[0000TTTT]	[1,1]	7 [00000111]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	8 [00001000]
[1,8] ✔	[0000TTTT]	[1,1]	7 [00000111]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	8 [00001000]
[1,8] ✔	[0000TTTT] ✔	[1,1]	7 [00000111]

ASPECTS OF EXECUTION

VALUE BASED

Sampling

Data Range	Constant Bits	Population Count	Value
[32,32]	[00100000]	[1,1]	32 [00100000]
[32,33]	[0010000T]	[1,2]	33 [00100001]
[32,36]	[00100T0T]	[1,2]	36 [00100100]

Constraint checking

Data Range	Constant Bits	Population Count	Value
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	2 [00000010]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✔	8 [00001000]
[1,8] ✔	[0000TTTT] ✔	[1,1] ✘	7 [00000111]

ASPECTS OF EXECUTION

CONTROL-FLOW BASED

- *Sample* and *check* for simple control flow invariants
- Paths: sample and check value of a *branch history vector* and given program points
- Calls: sample and check ID of caller in a callee's header

OVERHEAD OF SYSTEM

	Overhead (factor over `-O1`)				
Benchmark	*Full*	*CF*	*CS*	*VB*	*Selective*
`bc`	10.8	2.5	1.0	8.6	3.2
`bzip2`	21.4	3.4	1.0	19.0	4.3
`compress`	8.5	2.2	0.9	7.5	4.4
`grep`	4.2	1.3	1.0	4.0	1.7
`gzip`	16.4	4.1	1.0	13.1	7.2
`jpeg`	29.8	3.1	1.0	27.7	4.2
`libpoppler`	9.8	0.8	1.0	9.2	0.9
`libtiff`	15.0	1.3	1.0	15.0	4.5
`libvorbis`	15.0	1.5	1.0	14.8	5.6
`tar`	1.1	1.0	1.0	1.1	1.0
`wc`	4.3	1.9	1.0	4.4	1.9

DETECTING SOFT ERRORS