Toward provenance-based security for configuration languages

Paul Anderson
James Cheney

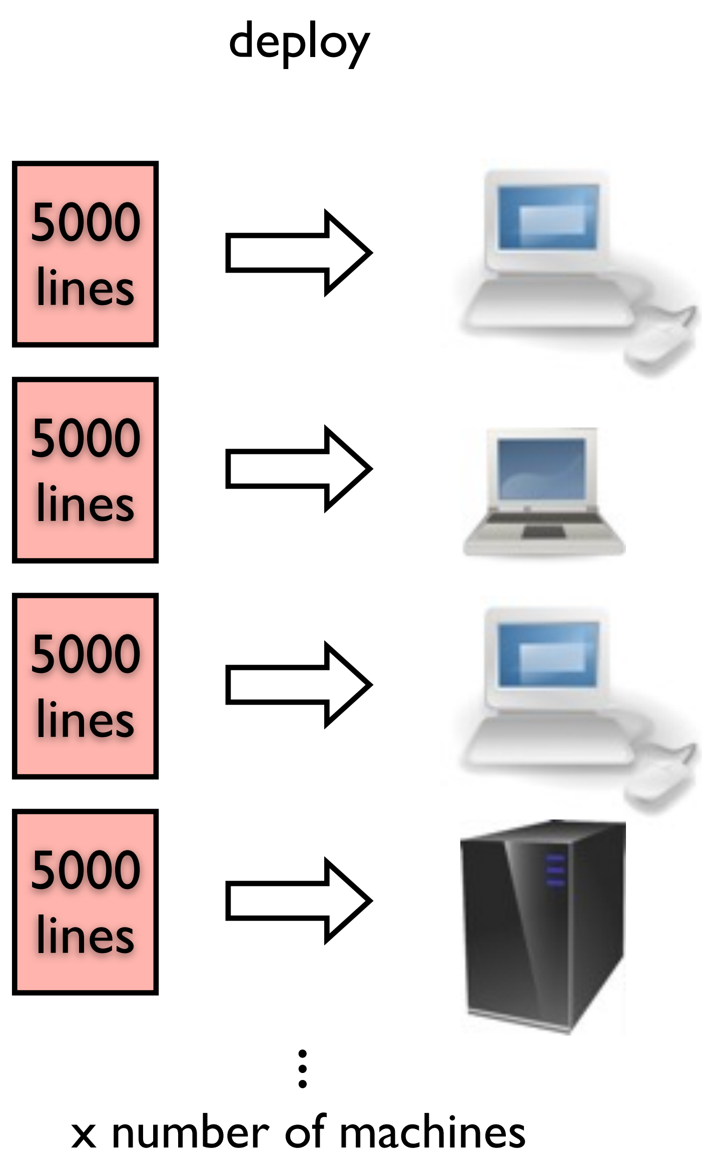
University of Edinburgh

TaPP, June 14, 2012

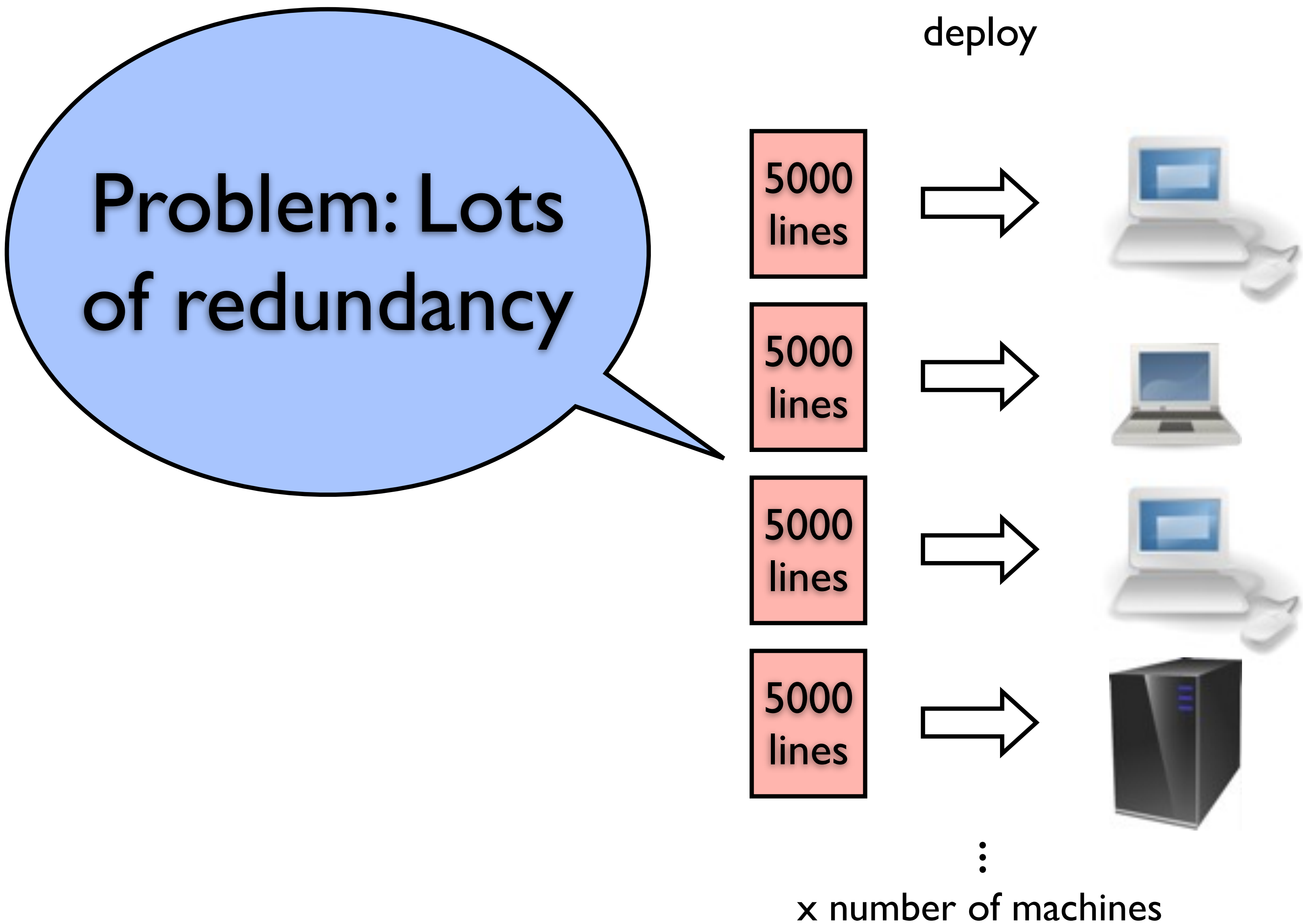
Configuration management

- Keeping machines updated / upgraded
- Keeping network access (firewalls, services) correctly configured
- Increasingly, declarative/high-level languages preferred
  - better maintainability
- LCFG (Edinburgh), Puppet, several other tools

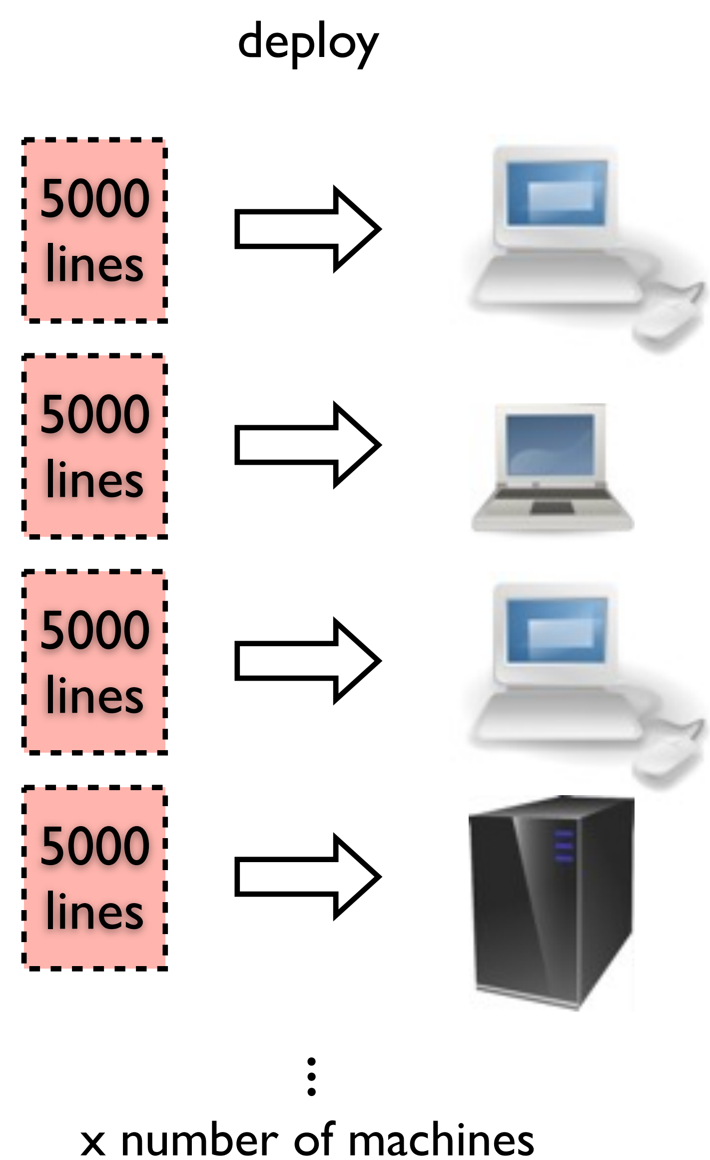
Declarative configurations



Declarative configurations

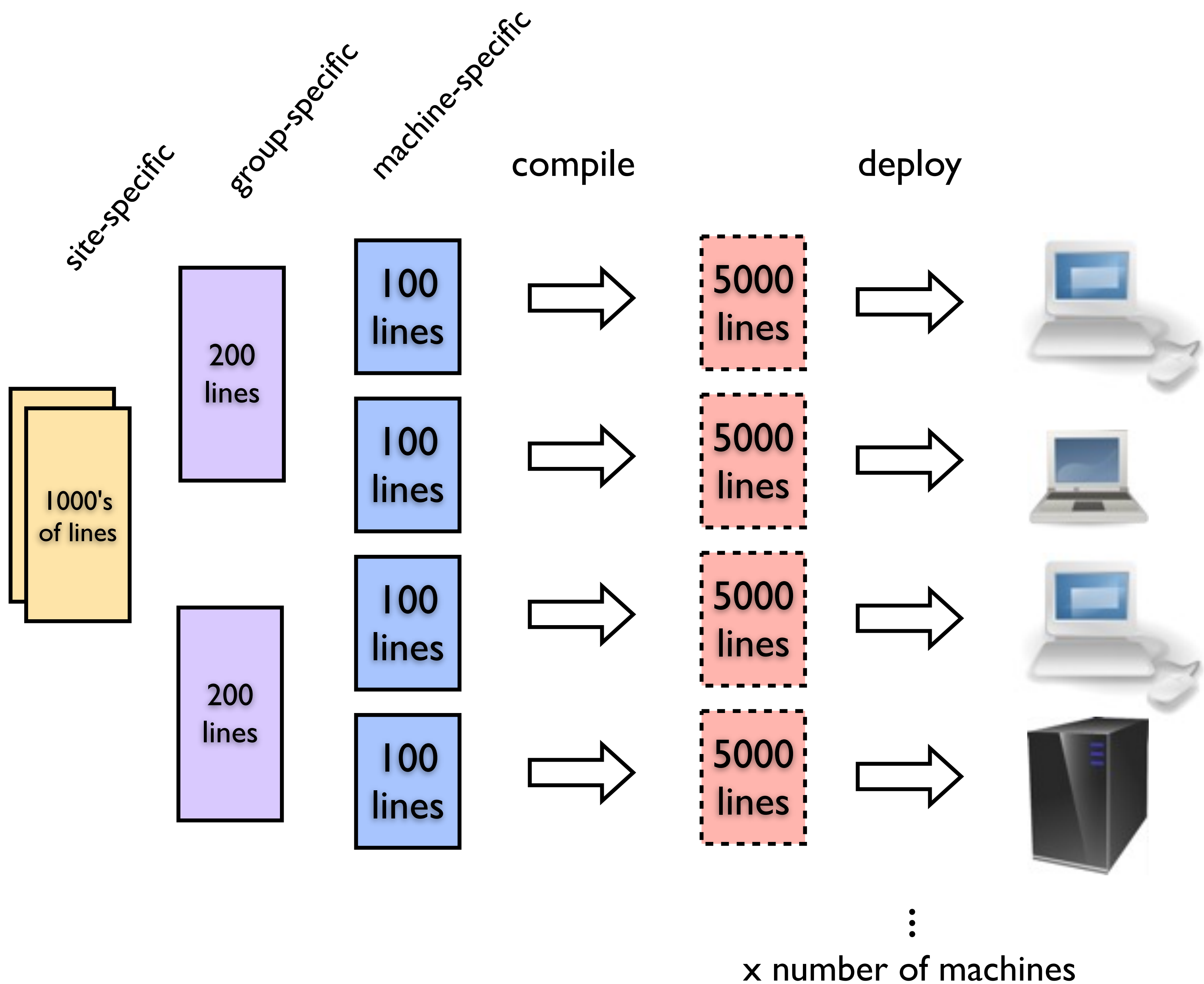


Smarter way

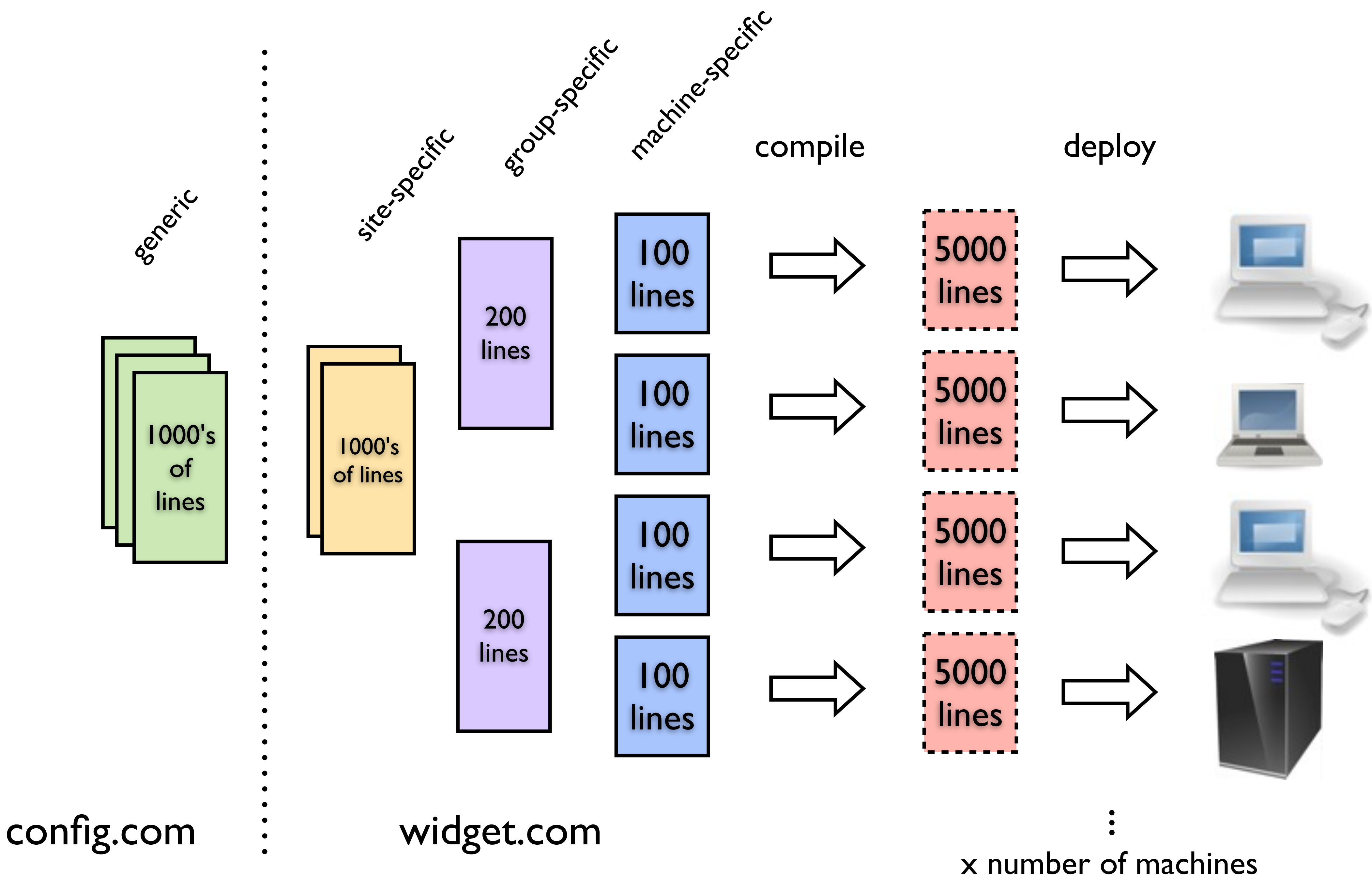


deploy
5000 lines
5000 lines
5000 lines
5000 lines
⋮
x number of machines

Smarter way


site-specific
group-specific
machine-specific
compile
deploy
1000's of lines
200 lines
200 lines
100 lines
100 lines
100 lines
100 lines
5000 lines
5000 lines
5000 lines
5000 lines
x number of machines

Smarter way


generic
site-specific
group-specific
machine-specific
compile
deploy
1000's of lines
1000's of lines
200 lines
200 lines
100 lines
100 lines
100 lines
100 lines
5000 lines
5000 lines
5000 lines
5000 lines
config.com
widget.com
x number of machines

Why is configuration management important?

- "In his case study on Linux system engineering in air traffic control, Stefan Schimanski showed how scalable Puppet really is and how it can guarantee reliable mass deployment of the Linux-based, **mission critical applications** needed in **air-traffic control centers**."
  - Linux Weekly News

Security

- Configuration management can lead to **vulnerabilities** in the overall system
  - even if individual components are secure
- Configurations often maintained in a **distributed** way
  - (across system/control boundaries)
- and compiled into (large) configuration files
- leading to potential for **mistakes** or **exploitation**

Example

config.com

```
class genericServer {
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Alice

widget.com

Example

config.com

```
class genericServer {

    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Alice

---

widget.com

```
class widgetServer isa genericServer {

     ...
}
```

Bob

Example

config.com

```
class genericServer {
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

widget.com

```
class widgetServer isa genericServer {
    ...
}
```

Bob

Prototype-based data (instance) inheritance

Example

config.com

```
class genericServer {

    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Prototype-based data (instance) inheritance

widget.com

```
class widgetServer isa genericServer {

    ...
}
```

Bob

```
class salesServer isa widgetServer {
    ...
}
```

Carol

Example

Prototype-based data (instance) inheritance

config.com

```
class genericServer {
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

widget.com

```
class widgetServer isa genericServer {
    ...
}
```

Bob

```
class salesServer isa widgetServer {
    ...
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

Example

config.com

```
class genericServer {
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Prototype-based data (instance) inheritance

widget.com

```
class widgetServer isa genericServer {
```

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

```
}
```

Example

config.com

```
class genericServer {
    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Alice

widget.com

```
class widgetServer isa genericServer {
    ...
}
```

Bob

```
class salesServer isa widgetServer {
    ...
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

Example

config.com

```
class genericServer {

    timeServer = ts@reliable.com
    ... 742 more parameters ...
}
```

Carol makes a temporary change...

widget.com

```
class widgetServer isa genericServer {

    ...
}
```

Bob

```
class salesServer isa widgetServer {
    timeServer = sales.widget.com
    ...
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

Example

config.com

```
class genericServer {
    timeServer = ts@unreliable.com
    ... 742 more parameters ...
}
```

Alice

```
class widgetServe  isa genericServer {
```

Bob

```
erver {
e.com
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

widge m

Meanwhile, Alice changes the default

Example

config.com

```
class genericServer {

    timeServer = ts@unreliable.com
    ... 742 more parameters ...
}
```

Alice

widget.com

```
class widgetServer isa genericServer {
```

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@sales.widget.com
    ...
}
```

Alice's change is masked by Carol's

Example

config.com

```
class genericServer {
    timeServer = ts@unreliable.com
    ... 742 more parameters ...
}
```

Alice

widget.com

```
class widgetServer isa genericServer {
    ...
}
```

Bob

```
class salesServer isa widgetServer {
  timeServer = ts@sales.widget.com
  ...
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

Example

config.com

```
class genericServer {

    timeServer = ts@unreliable.com
    ... 742 more parameters ...
}
```

widget.com

```
class widgetServer isa genericServer {

    ...
}
```

```
class salesServer isa widgetServer {
    ...
}
```

Carol

```
node serverA isa salesServer {
    ip = 1.2.3.4
    ...
}
```

Dave

Later Carol reverts the temporary change

Example

config.com

```
class genericServer {

    timeServer = ts@unreliable.com
    ... 742 more parameters ...
}
```

widget.com

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@unreliable.com
    ...
}
```

Suddenly things break, Carol gets the blame, but the real "culprit" is Alice

Workarounds

- Validation? (e.g. whitelists)
  - Validating final result will catch error earlier
  - But would not help identify cause
  - (and system could get into "stuck" state)
- Access control? [Vanbrabant et al. 2011]
  - Controlling effects of changes on final product can be unpredictable
  - Stuckness, inversion of privilege can result

Alternative

- Track provenance to understand flow of information through "compilation"
  - Change-history/provenance of source documents is usually available (e.g. SVN blame)
- Use to audit results
- or (perhaps) for provenance-aware access control

Challenges

- What is the right provenance model for (configuration language) security?
  - where-prov/"taint" tracking (ad hoc)
  - why-prov/dependency tracking (verbose)
  - override history? finer-grained traces?
- What is right tradeoff between exactness and utility?
- How do we measure / verify security guarantees involving provenance?

Challenges

- What is the right provenance model for (configuration language) security?

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@sales.widget.com
    ...
}
```

utility?

- How do we measure / verify security guarantees involving provenance?

Challenges

- What is the right provenance model for (configuration language) security?

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@sales.widget.com
    ...
}
```

Alice

utility?

- How do we measure / verify security guarantees involving provenance?

Challenges

- What is the right provenance model for (configuration language) security?

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@sales.widget.com
    ...
}
```

A B C Dave

utility?

- How do we measure / verify security guarantees involving provenance?

Challenges

- What is the right provenance model for

```
node serverA {
    ip = 1.2.3.4
    timeServer = ts@sales.widget.com
    ...
}
```

Ali

Carol

- How do we measure / verify security guarantees involving provenance?

Challenges

- What is the right provenance model for (configuration language) security?
  - where-prov/"taint" tracking (ad hoc)
  - why-prov/dependency tracking (verbose)
  - override history? finer-grained traces?
- What is right tradeoff between exactness and utility?
- How do we measure / verify security guarantees involving provenance?

Challenges #2

- Semantics of Puppet
  - largely data-description language
  - some object-orientation (inherit data, not code)
  - some functions/procedures, lists, ...
  - documentation focuses on examples, not corner cases
- other CLs have similar issues

Current work

- Currently investigating (arguably) simpler case of LCFG
  - Assign provenance to each line using `svn blame`
  - Adapt LCFG interpreter to provide finer-grained explanation of responsibility for each "part" of final config
- Complicated by heavy use of C pre-processor...
  - watch this space

Scale

- A "mature" configuration includes:
  - 13,764 lines
  - 7,244 statements
  - 268 unique files
    - written by 20-30 different people
    - 162 of which are included multiple times
- for one machine!

Related work

- Provenance security
  - Braun et al. (2007), Hasan et al. (2009) - systems perspective
  - Formal models: Cheney (CSF 2011), Acar et al. (POST 2012)
- Configuration languages/security
  - Change-based security for Puppet (Vanbrabant et al. 2011)
- Almost no work on **semantics** of config languages! (so maybe do that first...)

Conclusions

- Security is important for configuration languages
- Provenance may be useful for auditing and access control
- Future work:
  - understanding semantics of LCFG, Puppet or other languages
  - defining provenance models
  - defining security policies based on provenance