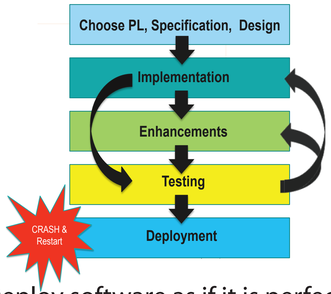
PASS
Perpetually Available & Secure Systems

Kathryn S McKinley
Emery Berger^, Michael Hicks*, Vitaly Shmatikov, Emmett Witchel
The University of Texas at Austin ^University of Massachussetts *University of Maryland

Goal: A transformative paradigm shift in deployment that that makes software more reliable, correct, and secure

Traditional Software Process

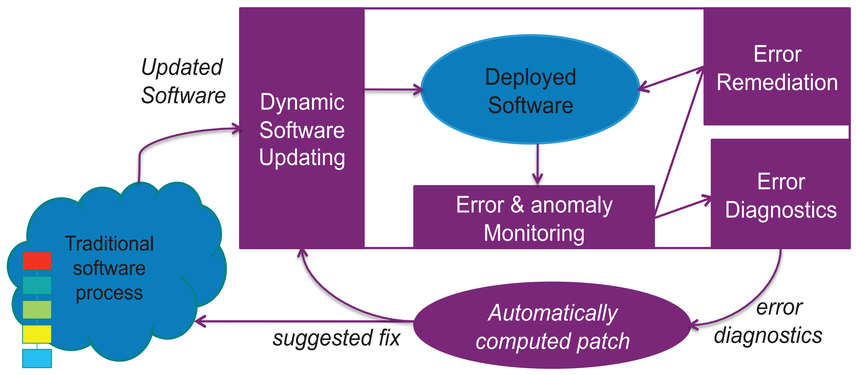


- Deploy software as if it is perfect
- Bugs & hardware failures crash software
- The US economy loses billions of dollars due to downtime & crashes
- Software failure compromises safety
- Ad hoc or no security & enforcement

PASS application deployment model: do not crash

- no downtime for errors or feature enhancements
- **anticipate**: monintor for hardware failiure & software bugs
- **diagnose**: combine dynamic & static error analysis
- **tolerate**: errors through data repair & code redirection
- **fix**: generate patch combining static analysis w/dynamic analysis & tolerance results
- **deploy with dynamic software update**: fix & add features while the program executes

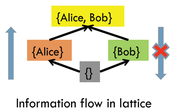
PASS Runtime System



PASS Security Model

DIFC: Distributed Information Flow Control
- Label data with secrecy and integrity
- Specify policies on data access by principals (threads)

PASS Enforcement: Laminar language, VM, & OS
- Security region programming model
- Fine grain access flow control
- Whole system enforcement of fine grain policiies



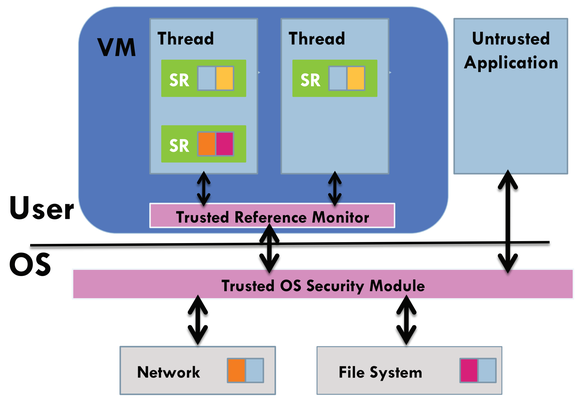
Information flow in lattice

Calendar	Monday	Tuesday
Alice	Watch game	Office work
Bob	Free	Meet doctor

```
Calendar cal;  // has label {Alice, Bob}
secure(new Label(Alice, Bob)){
    Calendar a = readFile("alice.cal");
    Calendar b = readFile("bob.cal");
    cal.addDates(a, b);
    Date d = cal.findMeeting();
... }
```

Can read Alice & Bob's data
Add to common calendar
Find common meeting time

PASS Security Enforcement



Memory Leak Example

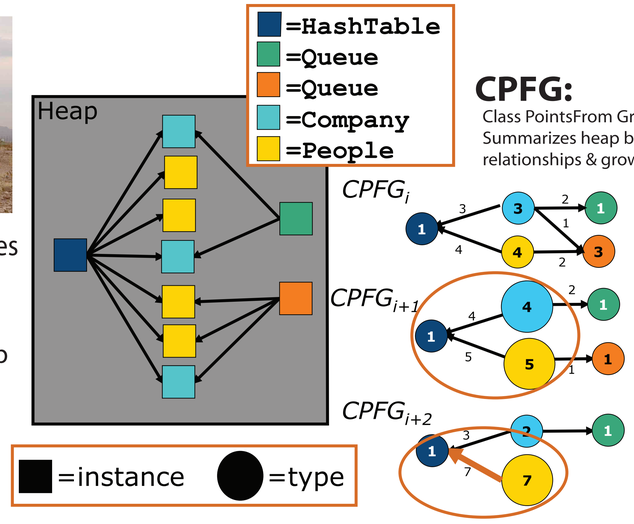
Darpa unmanned vehicle challenge

- Leak: past obstacles remained reachable
- Crashed in 40 minutes during training
- The plan for competition: Restart! every 40 minutes
- Reality: more obstacles, crash after 20 minutes

Summarize Heap

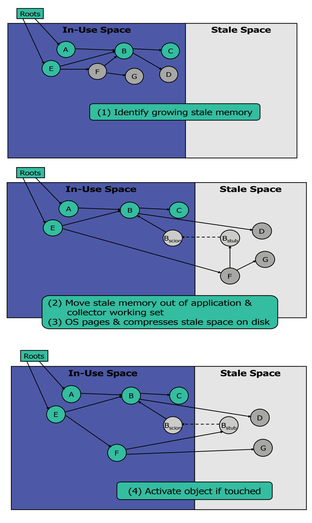
- Garbage collector observes heap
- CPFG: class points from graph
- Compare CPFGs to find heap growth in objects & references
- Report **Class Slice**

Leak Diagnosis



CPFG:
Class PointsFrom Graph
Summarizes heap by class relationships & growth

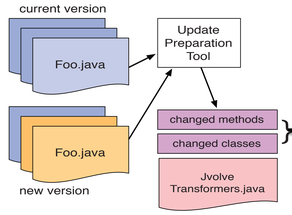
Leak Tolerance



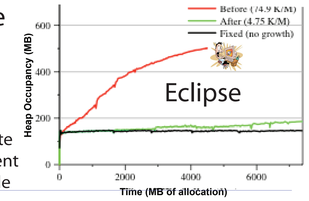
Code & Heap Correction

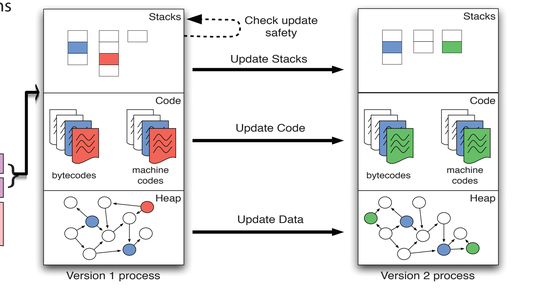
- Generate fix from dynamic diagnosis & tolerance results combined with static analysis

Update code with fix while executing:
- New code and object transformers specify update to code & data
- Expanded VM services analyze update and executable, time & apply update
- VM scheduler that finds & creates DSU safe point with on-stack-replacement
- VM compiler analzes code and determines effected source & machine code
- Garbage collector deletes, adds, transforms objects in the heap



Dynamic Software Updating





PASS DSU: Dynamic Software Update Process