Dynamic Software Updating
Definition

Program = tuple (q,P)
q = state
P = code

DSU transforms (q,P) to (q',P')
state q must be transformed into representation P' expects.

State transformer S
DSU transforms (q,P) to (S(q), P')

Update is valid iff (S(q), P') can be reduced to a (q,P')
that is reachable from start point (q_{init}, P')
static int num = 0;
int f(int a, int b) {
    num++;
    return a + b;
}

Figure 1: A file $f$

new version $f'$:
static int num = 0;
int f(int a, int b) {
    num++;
    return a * b;
}

state transformer $S$
void $S$ () {
    $f'$::num = $f$::num;
}

Figure 2: Dynamic patch for $f$: $(f', S)$
Motivation

- Software must be updated.
- Non-stop systems
- Non-critical systems?
- Update cost
DSU Properties

- **Flexibility**
  - Any part of a running system should be updatable

- **Correctness**
  - Minimize risk of error

- **Ease of use**
  - Low complexity
  - Automated patch generation

- **Low Overhead**
  - Performance
Existing approaches

- No existing general-purpose updating system meets all the criteria
- WHAT, WHEN, HOW - problems
- Time constraint vs overhead
Techniques

Type Wrapping

```
struct T {
    int x;
    int y;
};

struct __T1 {
    int x;
    int y;
};

struct T {
    unsigned int version;
    union { struct __T1 data;
        char padding[N]; } udata;
};
```
Techniques(2)

Function Indirection

```c
void foo(int x) {
  ...
}

void bar() {
  foo(1);
}

void foo_v0(int x) {
  ...
}

void * foo_ptr = &foo_v0;

void bar() {
  (*foo_ptr)(1);
}
```
Techniques(3)

- Loop extraction
- Static code analysis (Updateability Analysis)
- Type Transformers
- State Transformers
- Verification of DSU through testing
Questions ?