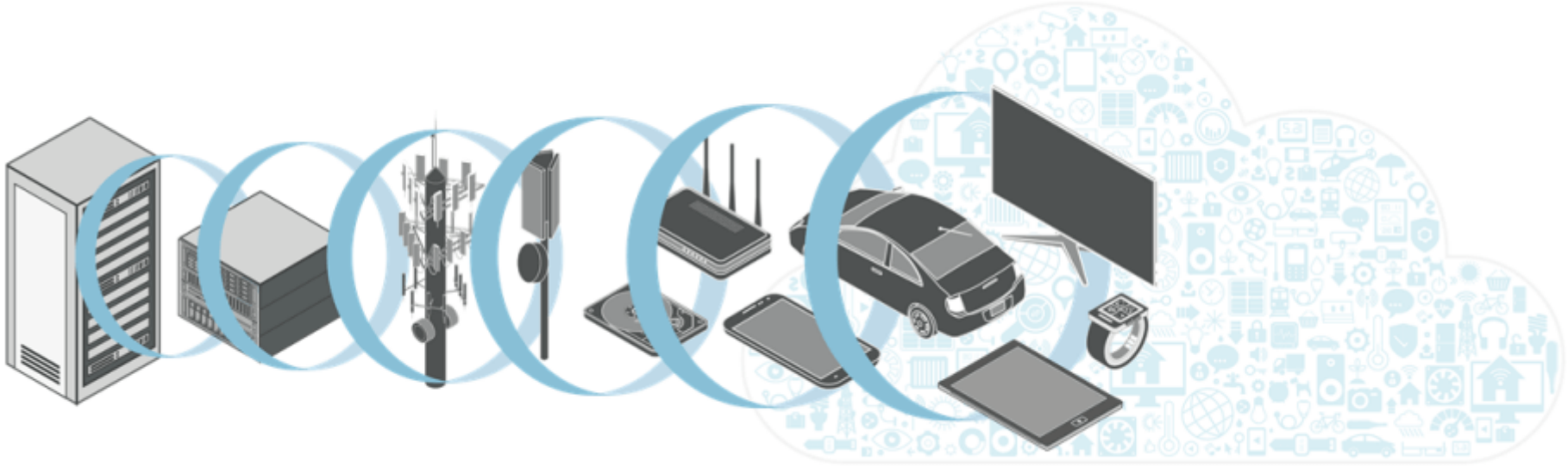
Optimization Goals



Performance ⟷ Code Size

Primary Goal

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;


  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Match

```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;

  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;

  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Match

```
void merged(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;

  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Mismatch

```
void merged(bool func_id,
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(func_id)
    if(quantum_objcode_put(COND_PHASE, control, target))
      return;
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;

  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Mismatch

```
void merged(bool func_id,
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(func_id)
    if(quantum_objcode_put(COND_PHASE, control, target))
      return;
  float var = (func_id)?pi:(-pi);
  z = quantum_cexp(var / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Motivating Example

```
void quantum_cond_phase(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(quantum_objcode_put(COND_PHASE, control, target))
    return;
  z = quantum_cexp(pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

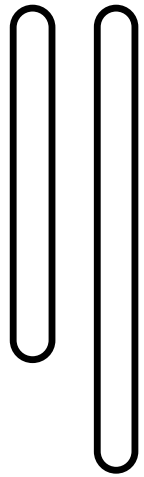
```
void quantum_cond_phase_inv(
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;


  z = quantum_cexp(-pi / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

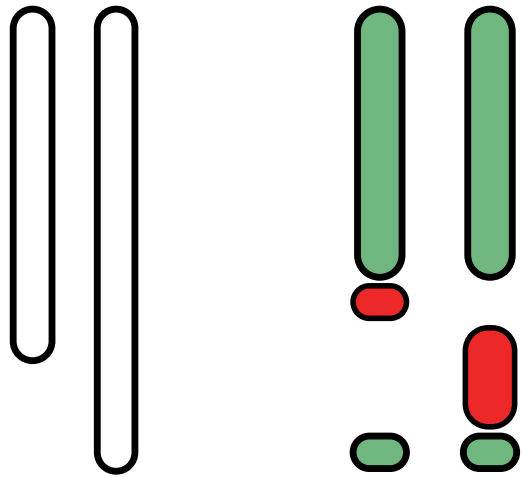
```
void merged(bool func_id,
int control, int target, quantum_reg *reg){
  int i;
  COMPLEX_FLOAT z;
  if(func_id)
    if(quantum_objcode_put(COND_PHASE, control, target))
      return;
  float var = (func_id)?pi:(-pi);
  z = quantum_cexp(var / (1 << (control - target)));
  for(i=0; i<reg->size; i++) {
    if(reg->node[i].state & (1 << control)) {
      if(reg->node[i].state & (1 << target))
        reg->node[i].amplitude *= z;
    }
  }
  quantum_decohere(reg);
}
```

Key Insight

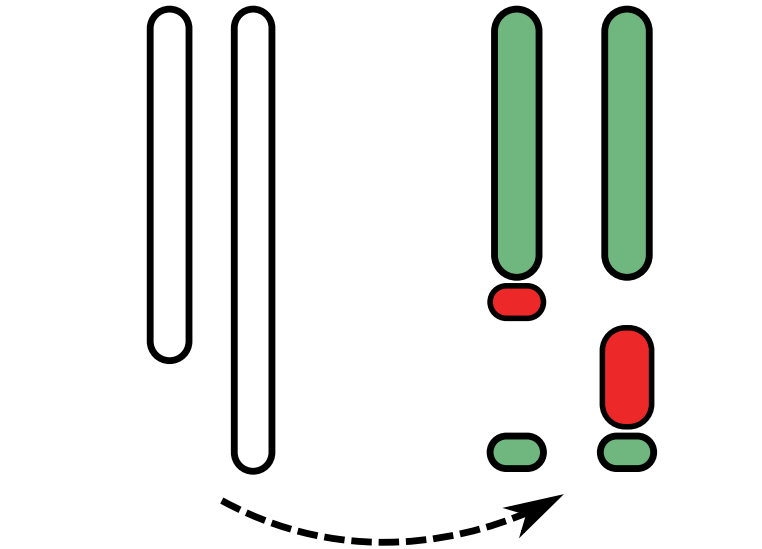
Key Insight

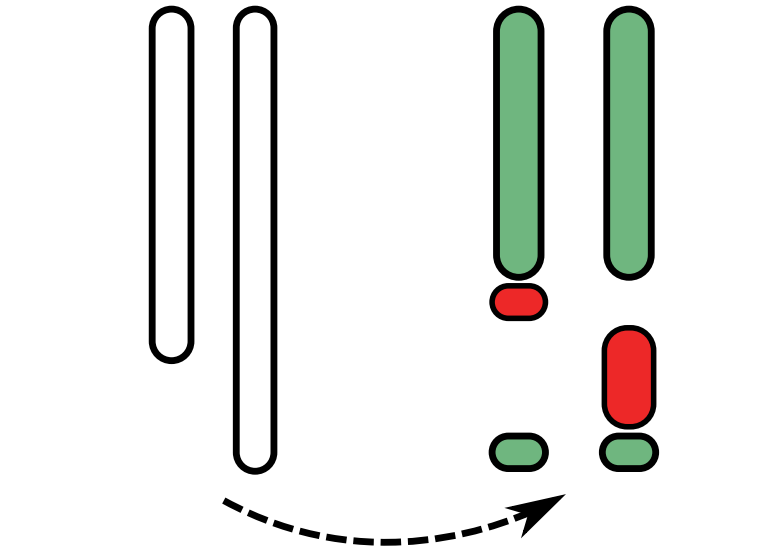


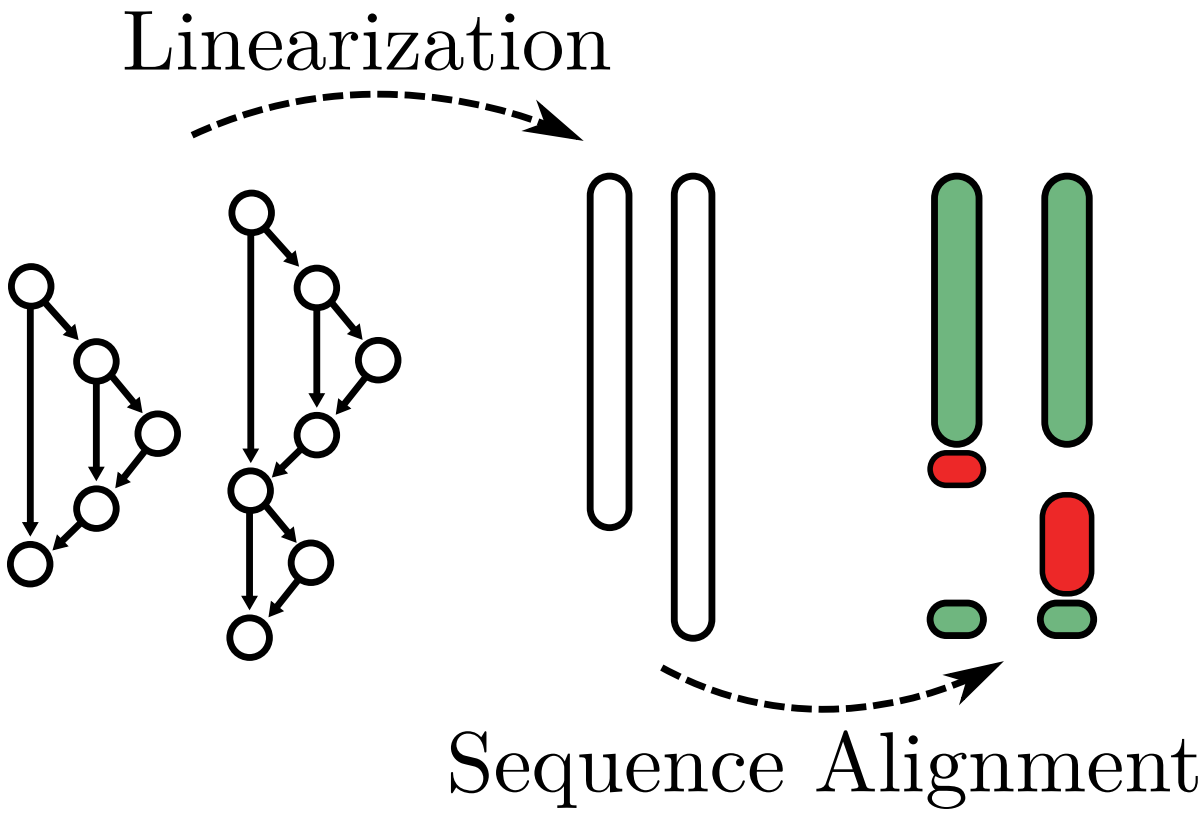
Key Insight

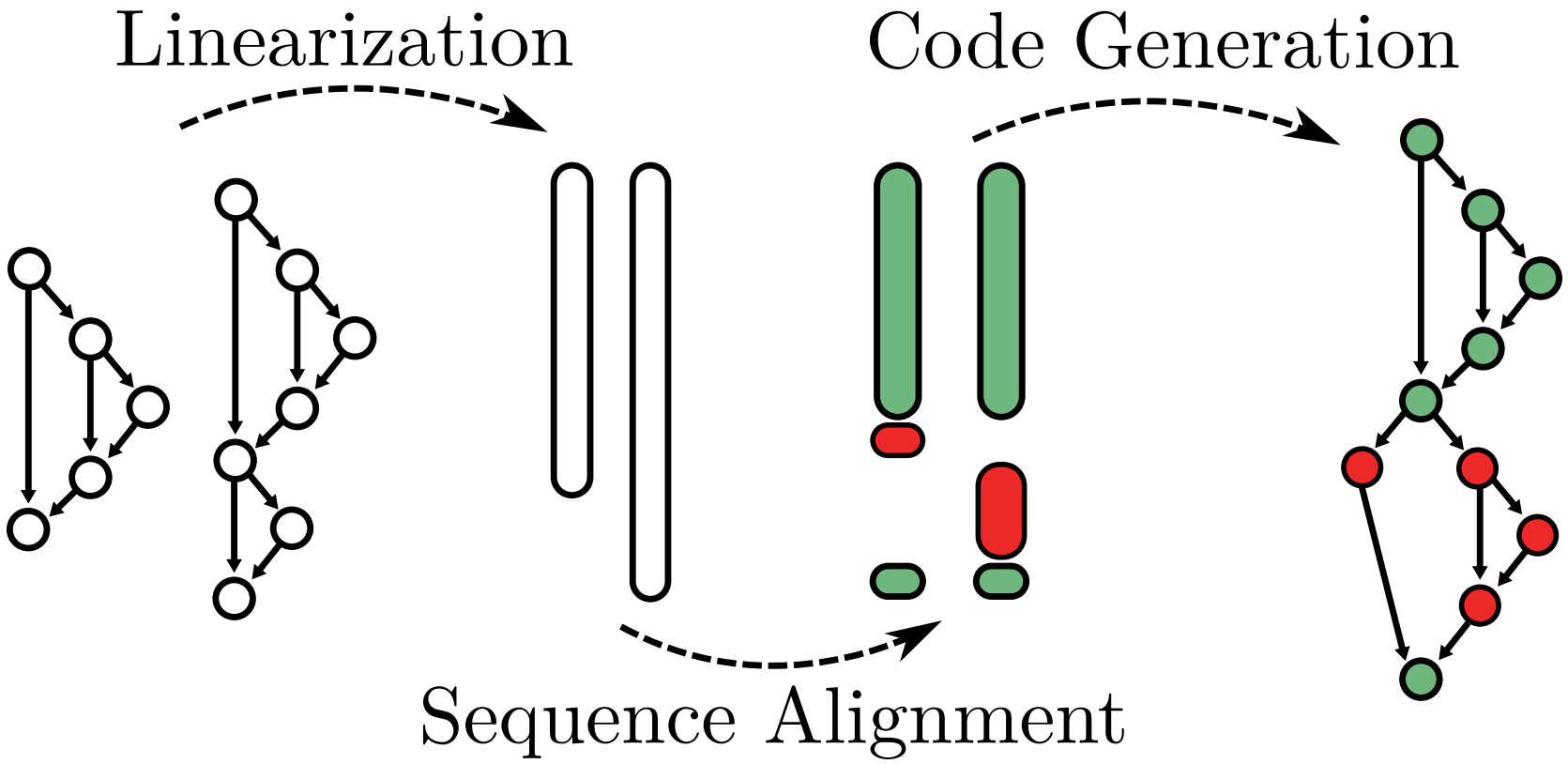


Key Insight

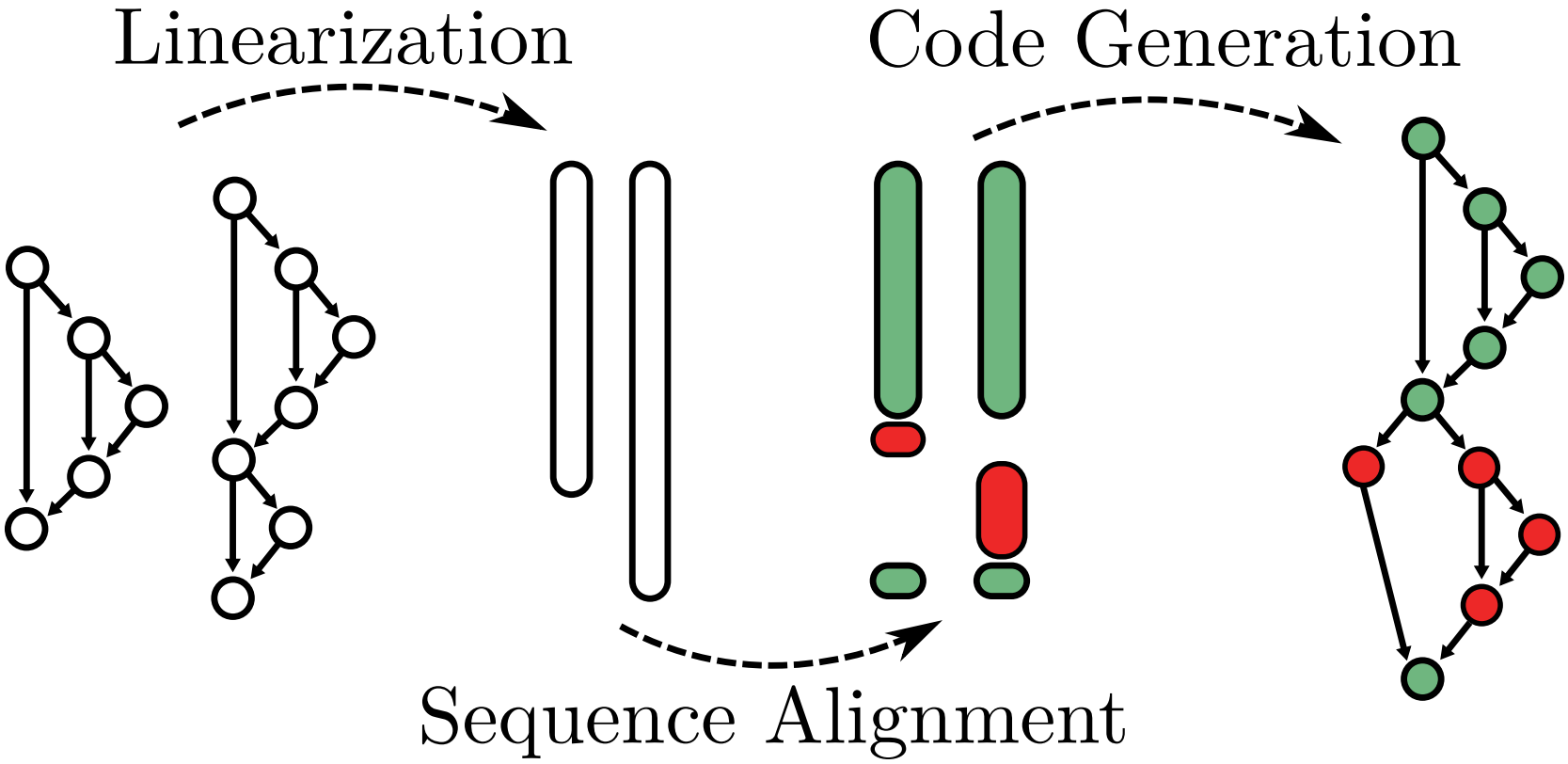



Sequence Alignment


Linearization
Sequence Alignment


Linearization
Code Generation
Sequence Alignment

Our Technique



Our Technique

Our Technique

```
double ExponentialRandom(void) {
  double x;
  do {
    x = sre_random();
  } while (x == 0.0);
  return -log(x);
}
```

Our Technique

```
double ExponentialRandom(void) {
  double x;
  do {
    x = sre_random();
  } while (x == 0.0);
  return -log(x);
}
```

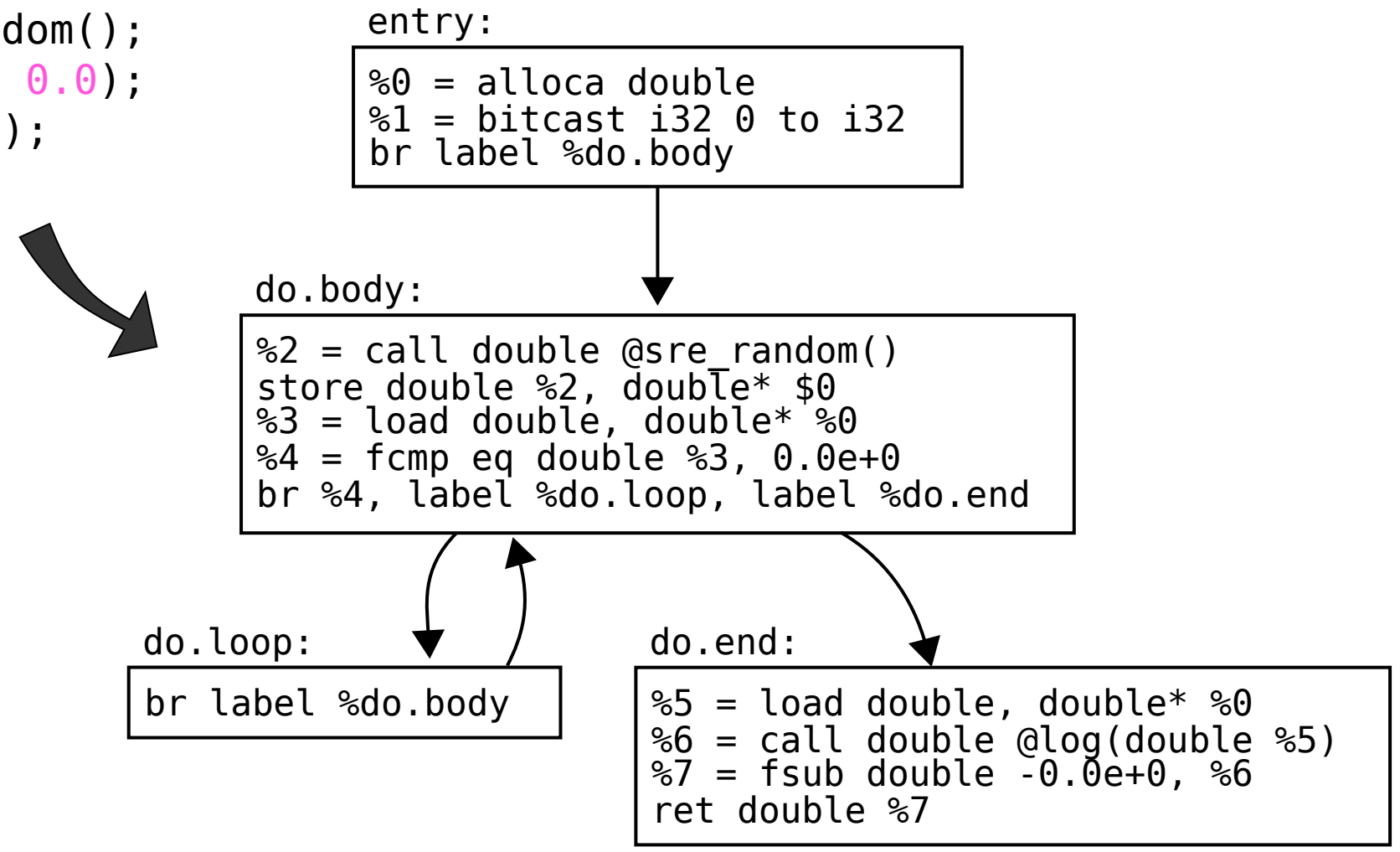
```
double sre_random_positive(void) {
  double x;
  do {
    x = sre_random();
  } while (x == 0.0);
  return x;
}
```

Our Technique

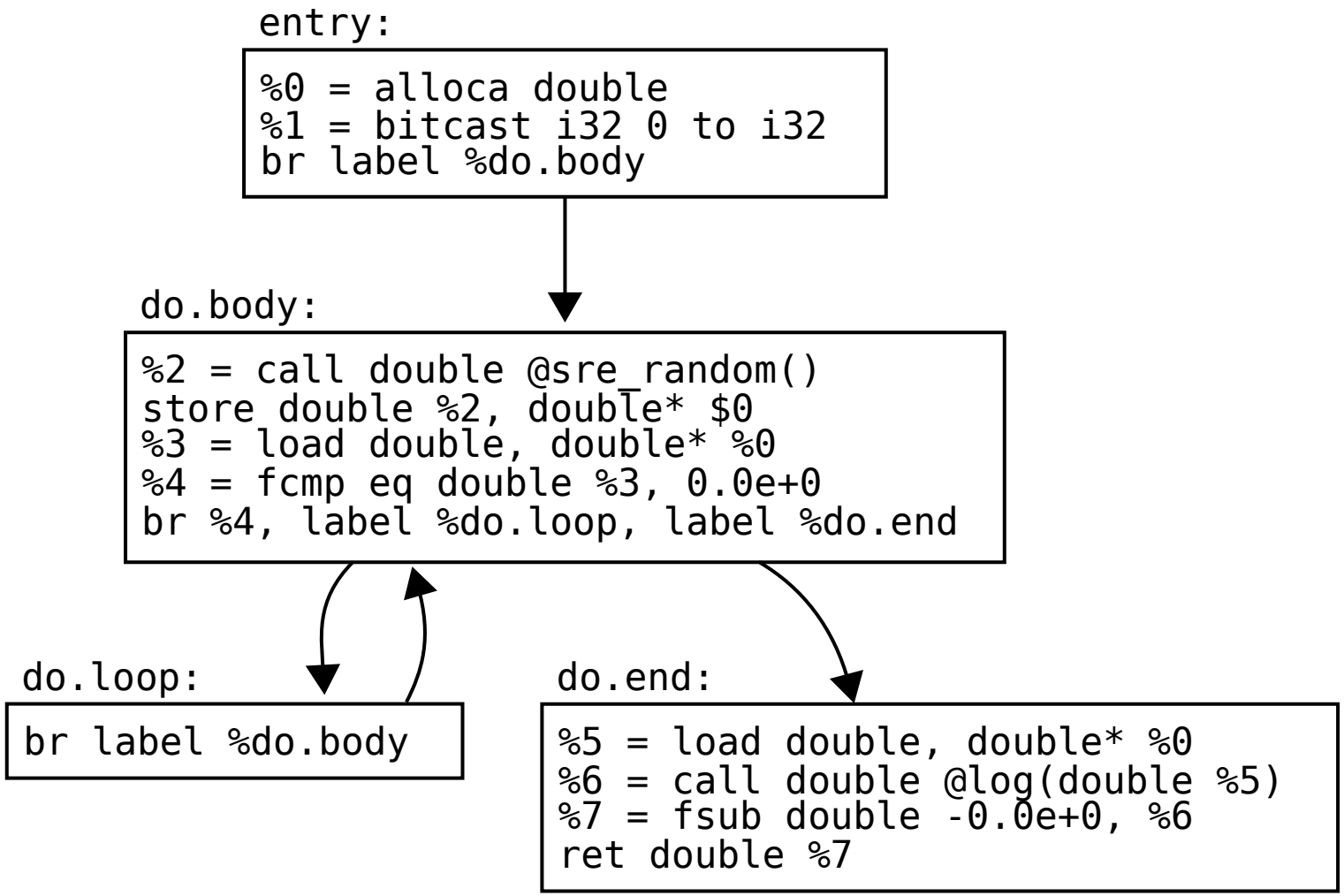
```
double ExponentialRandom(void) {
  double x;
  do {
    x = sre_random();
  } while (x == 0.0);
  return -log(x);
}
```

Our Technique

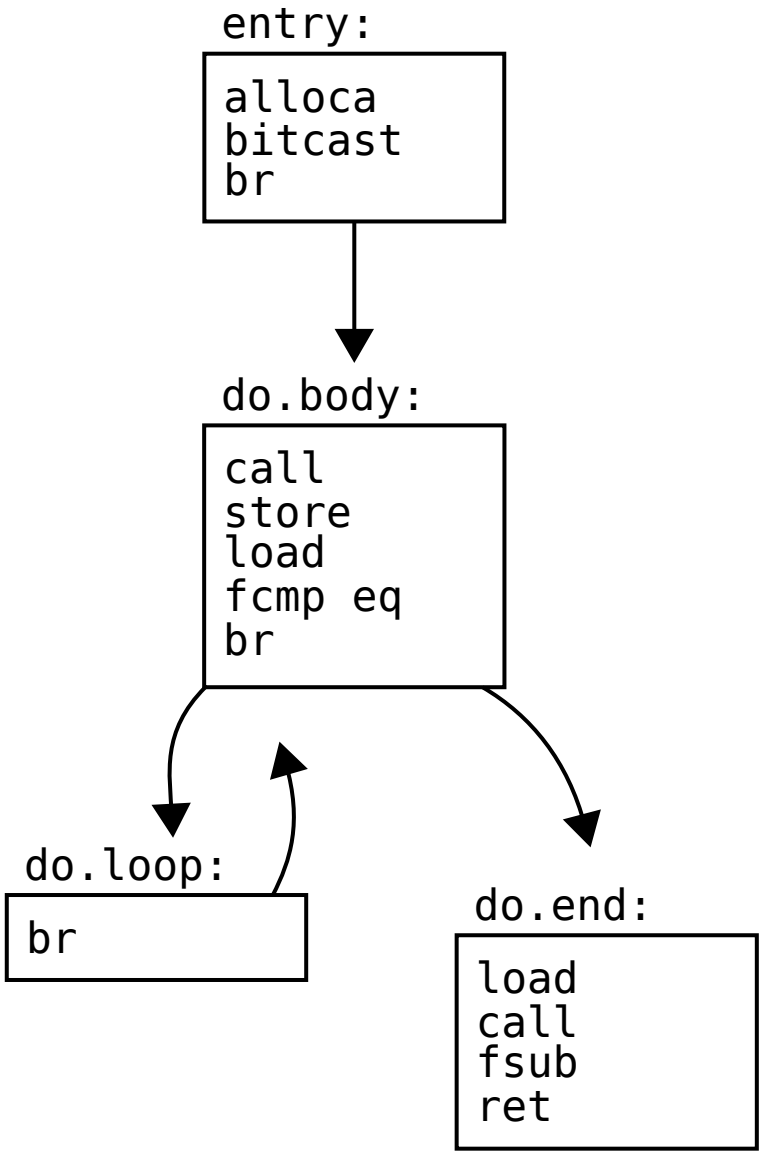
```
double ExponentialRandom(void) {
  double x;
  do {
    x = sre_random();
  } while (x == 0.0);
  return -log(x);
}
```



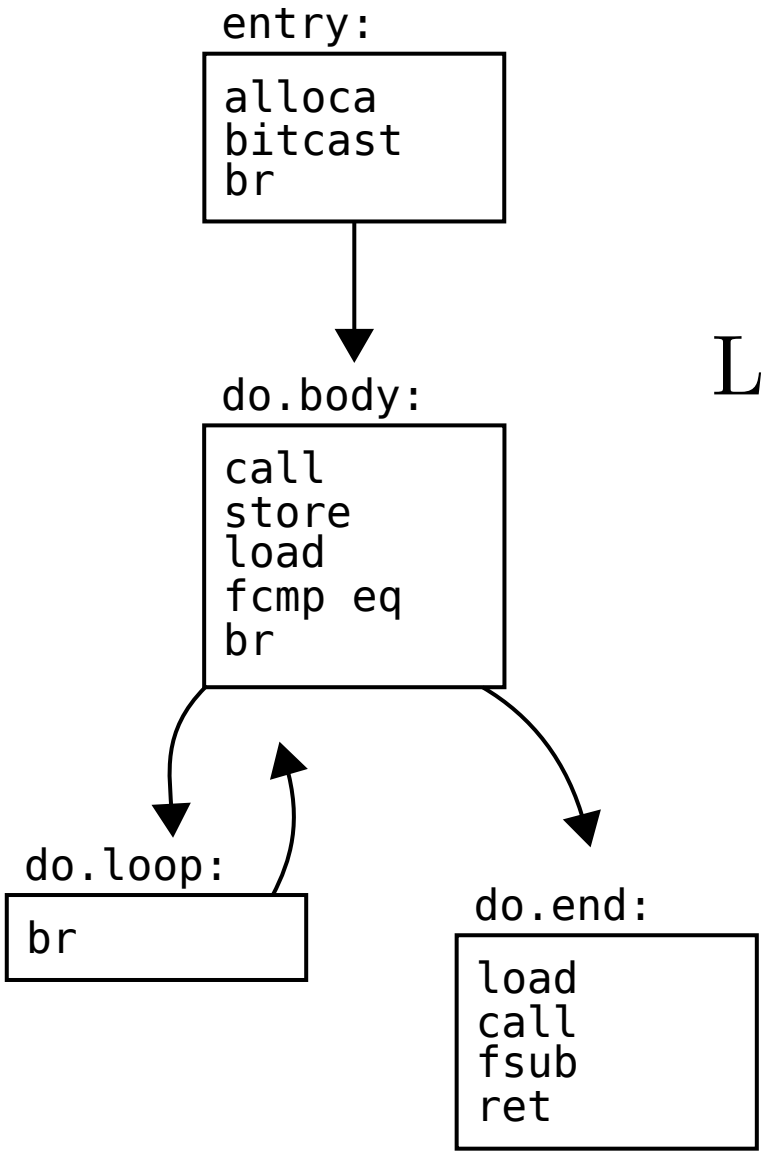
Our Technique



Our Technique

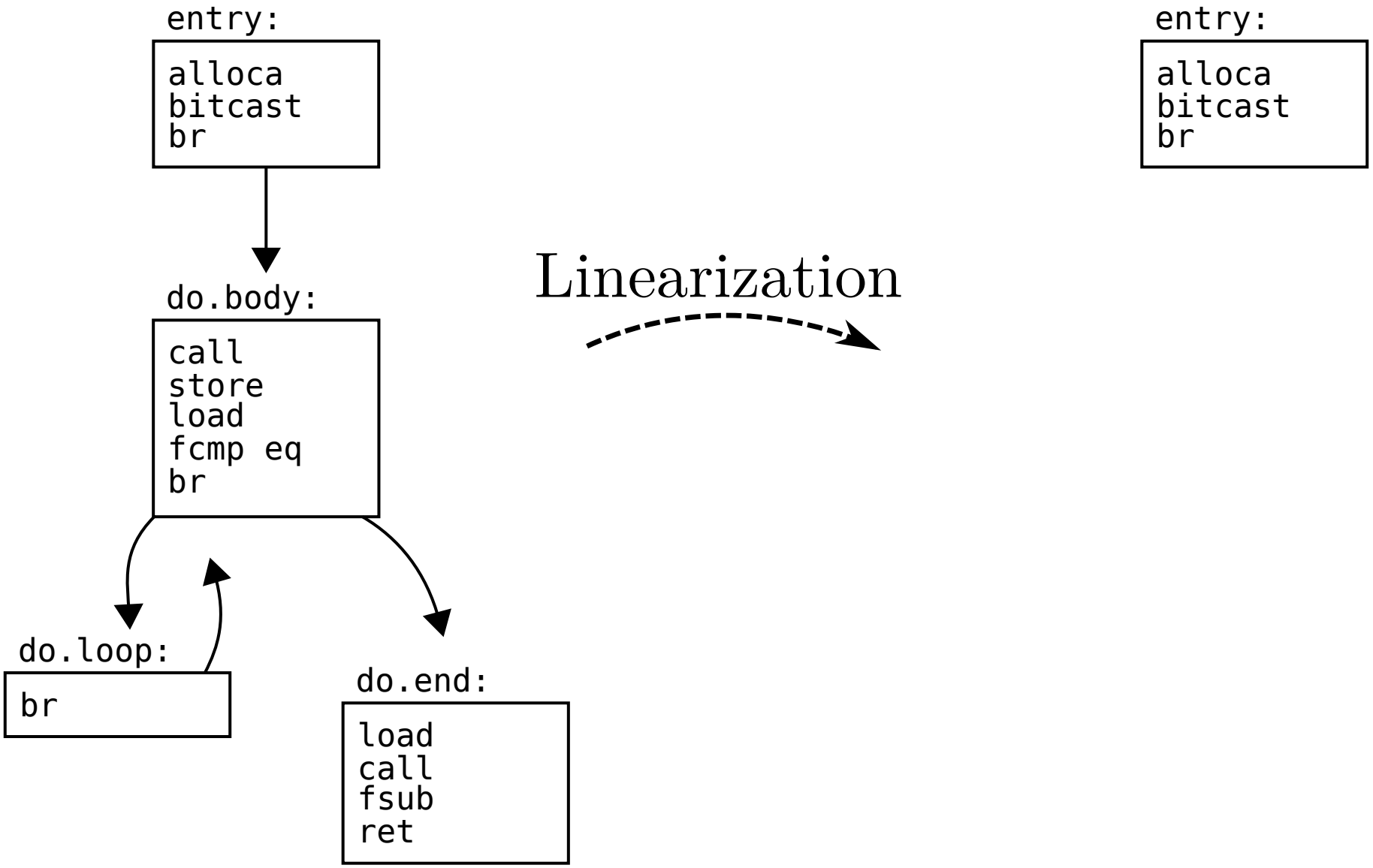


Our Technique

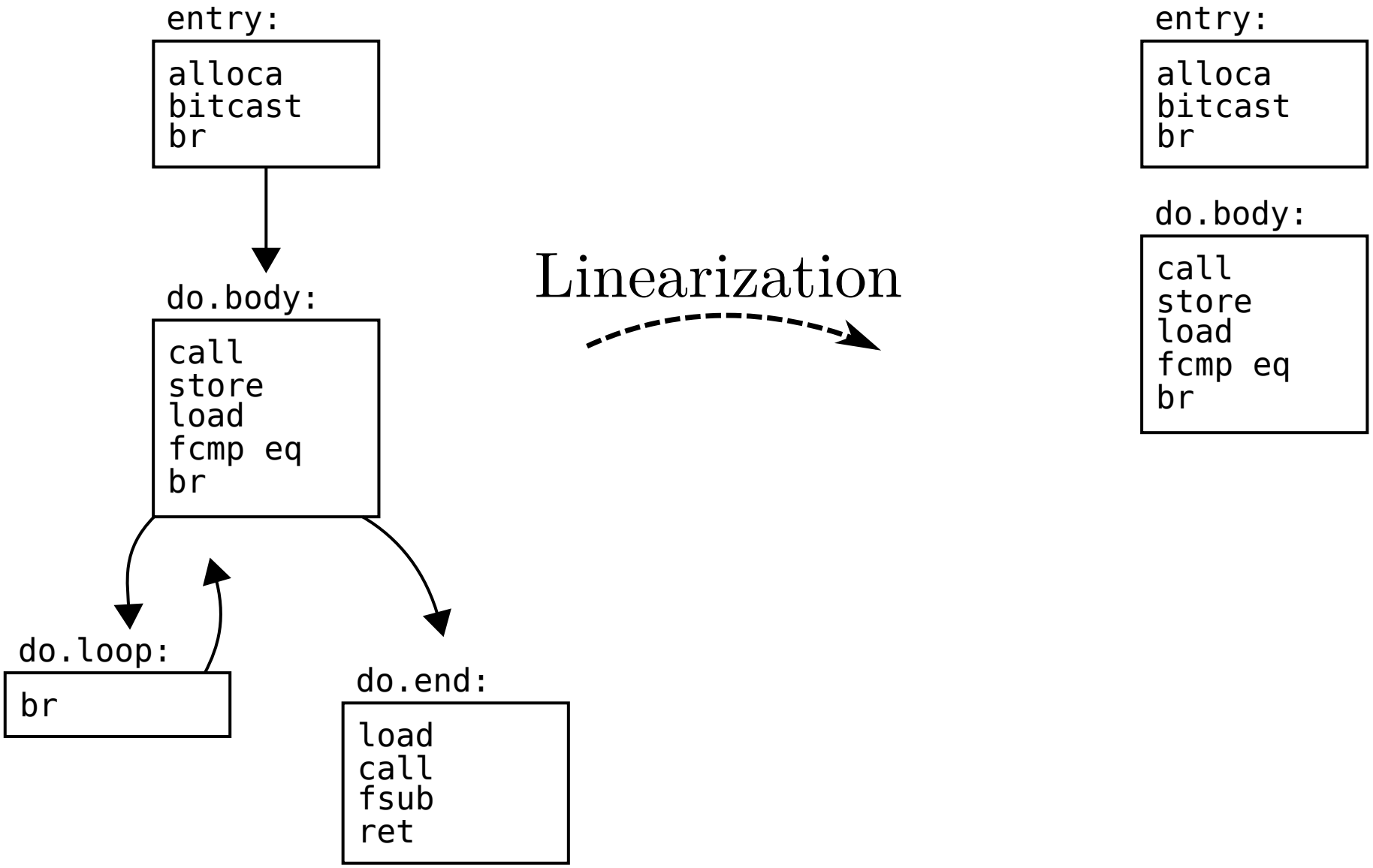


Linearization

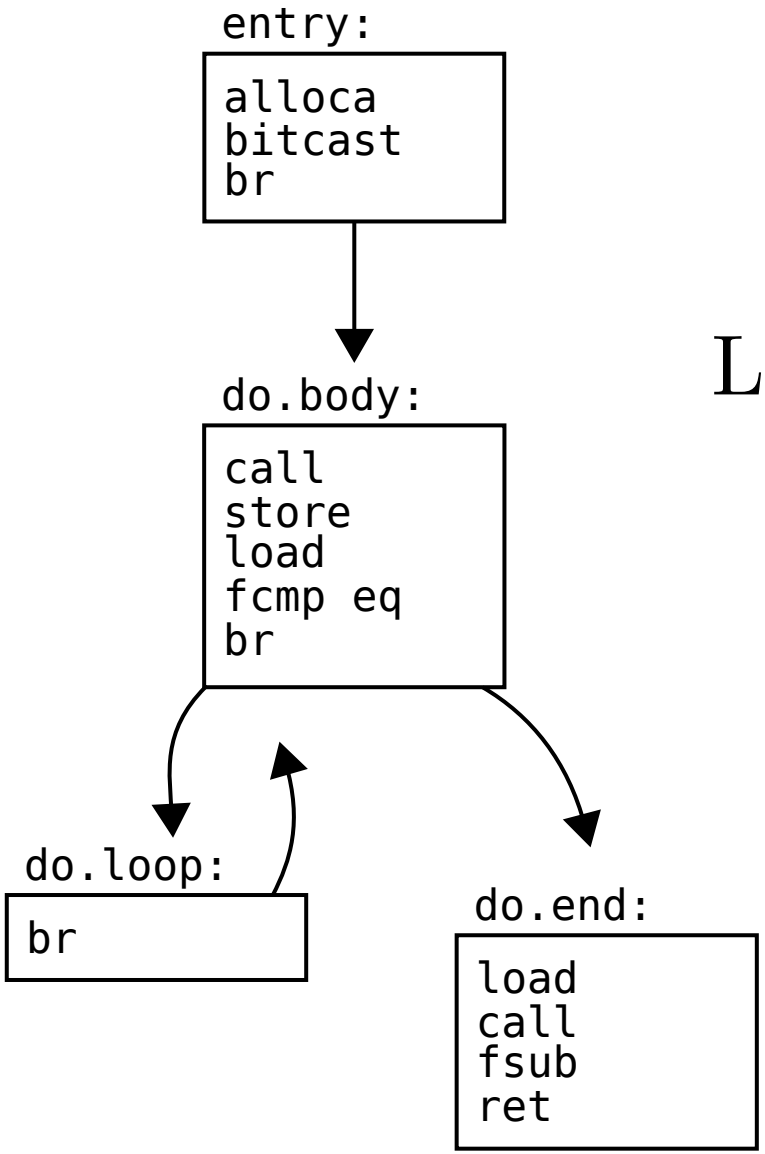
Our Technique



Our Technique

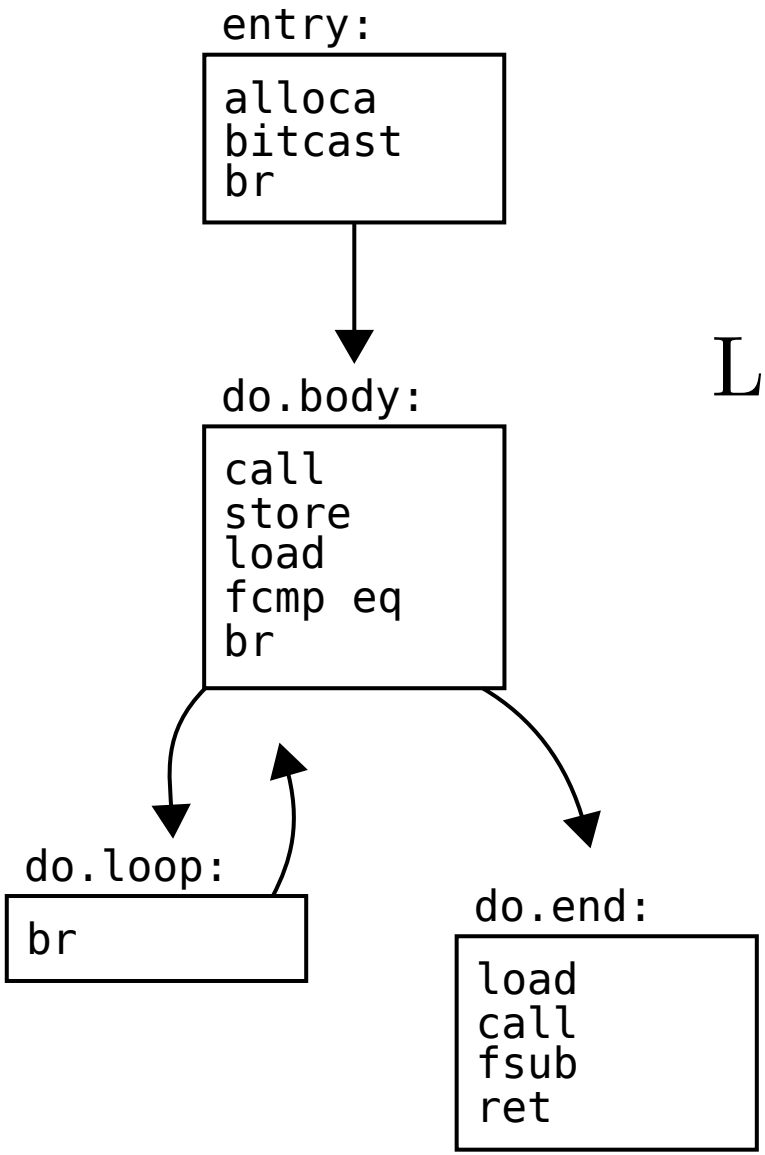


Our Technique

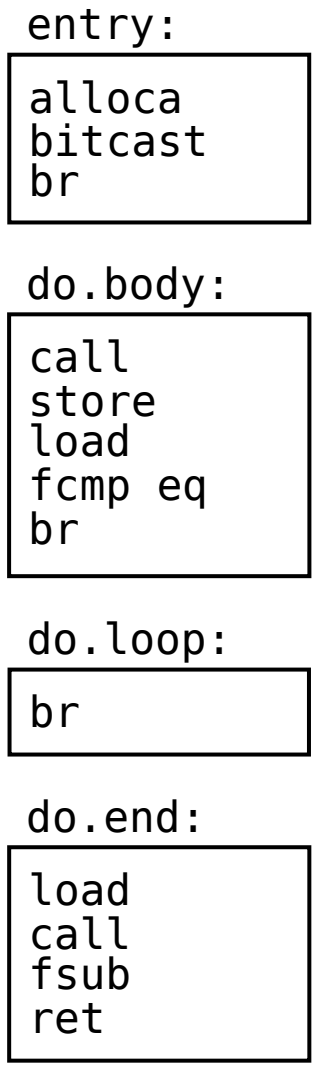


Linearization

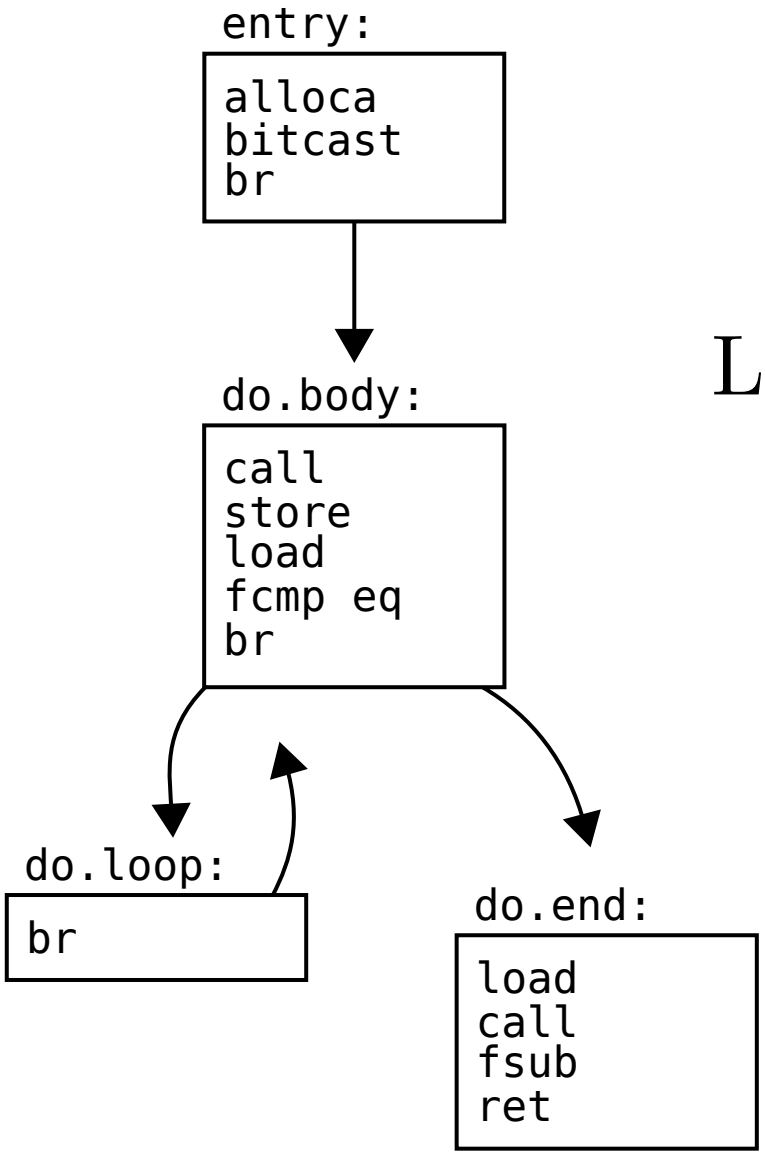
Our Technique



Linearization



Our Technique

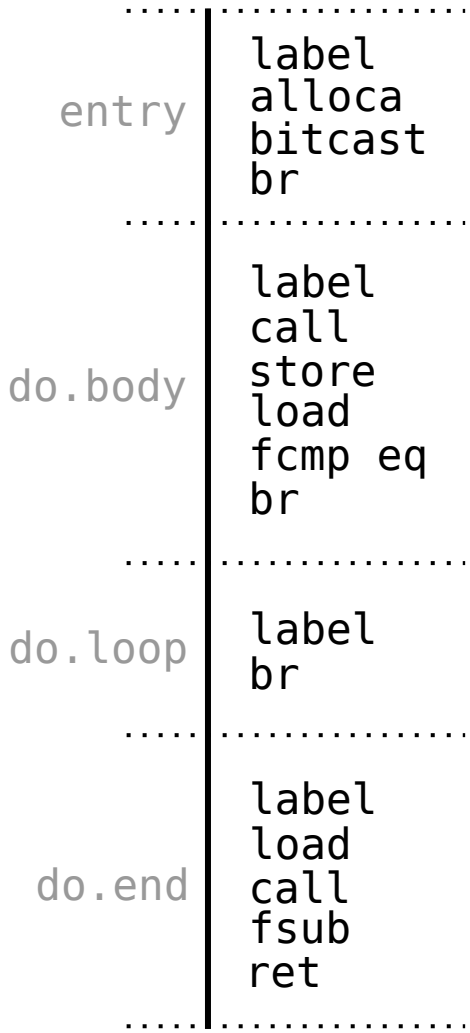


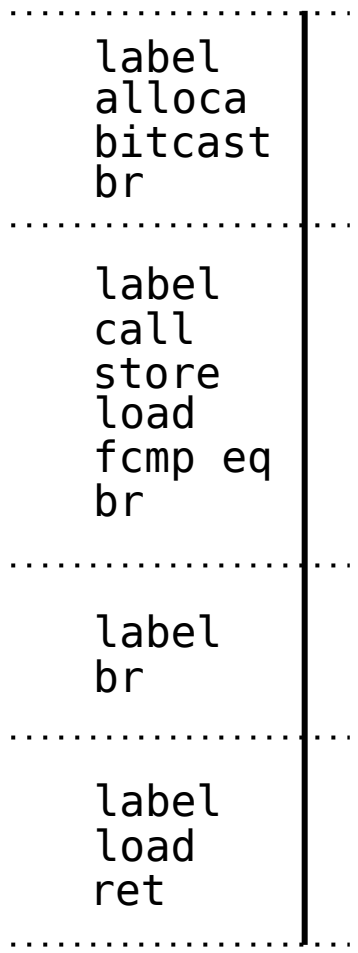
Linearization

entry	label alloca bitcast br
do.body	label call store load fcmp eq br
do.loop	label br
do.end	label load call fsub ret

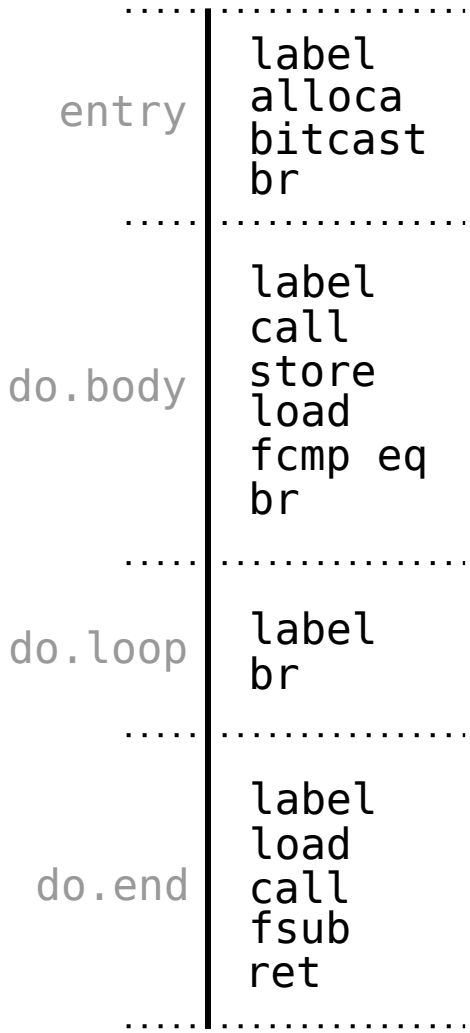
Our Technique

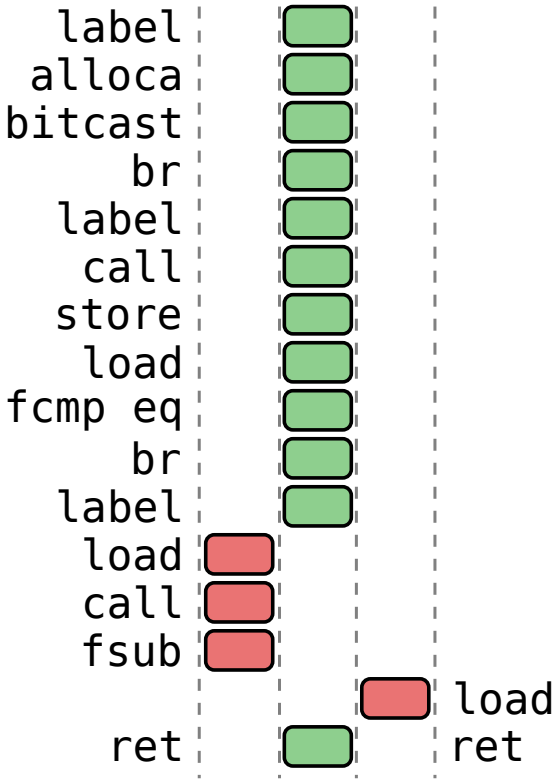
Our Technique

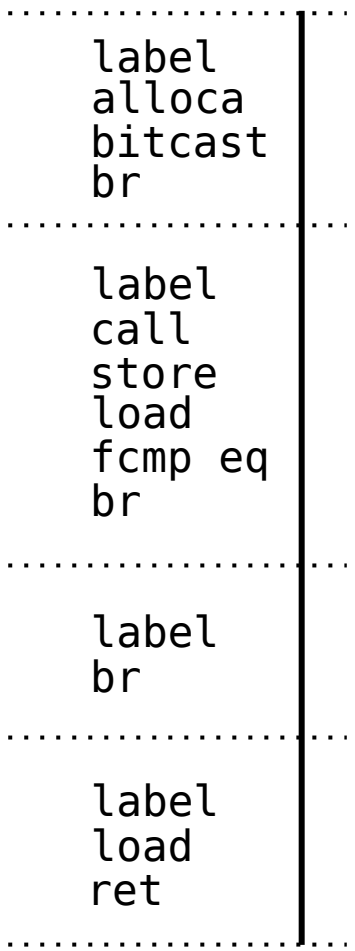




Our Technique

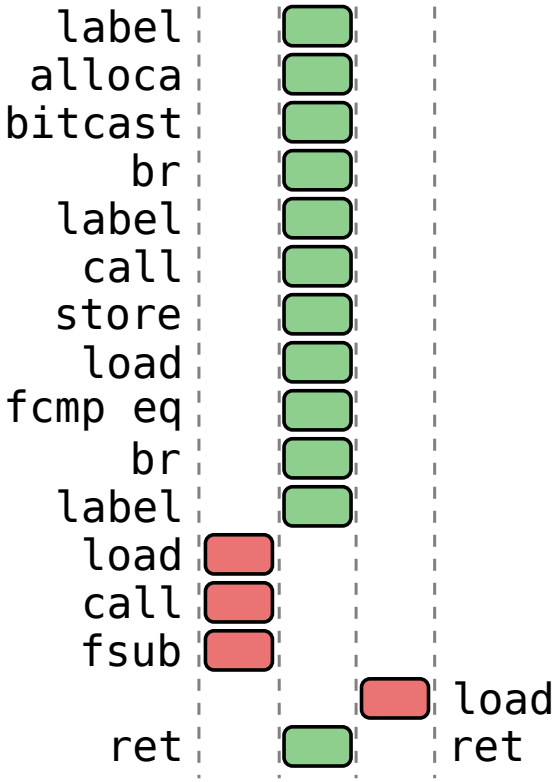




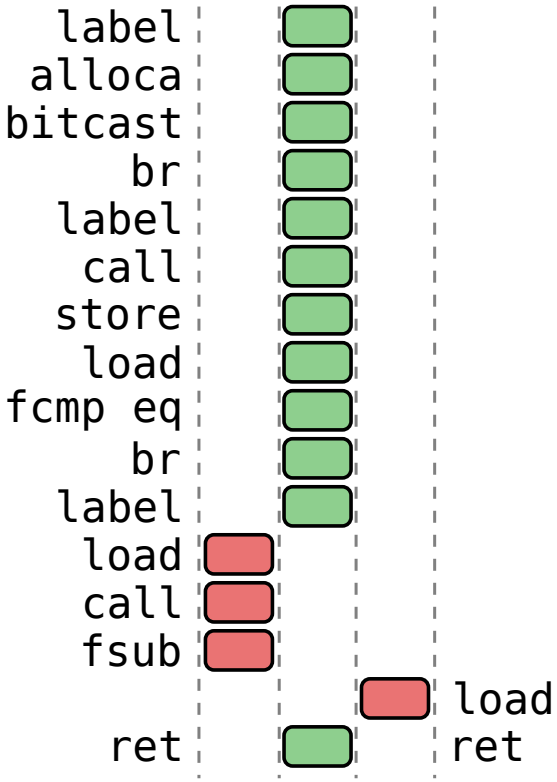


Sequence Alignment

Our Technique

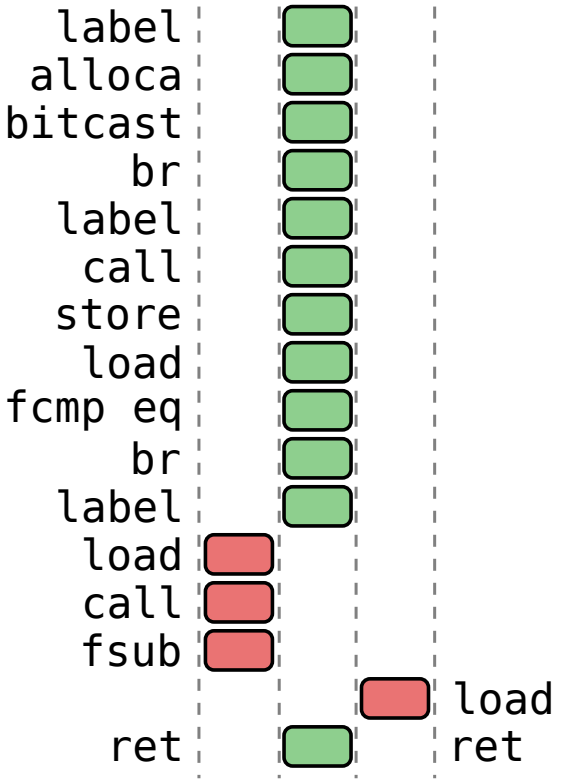


Our Technique

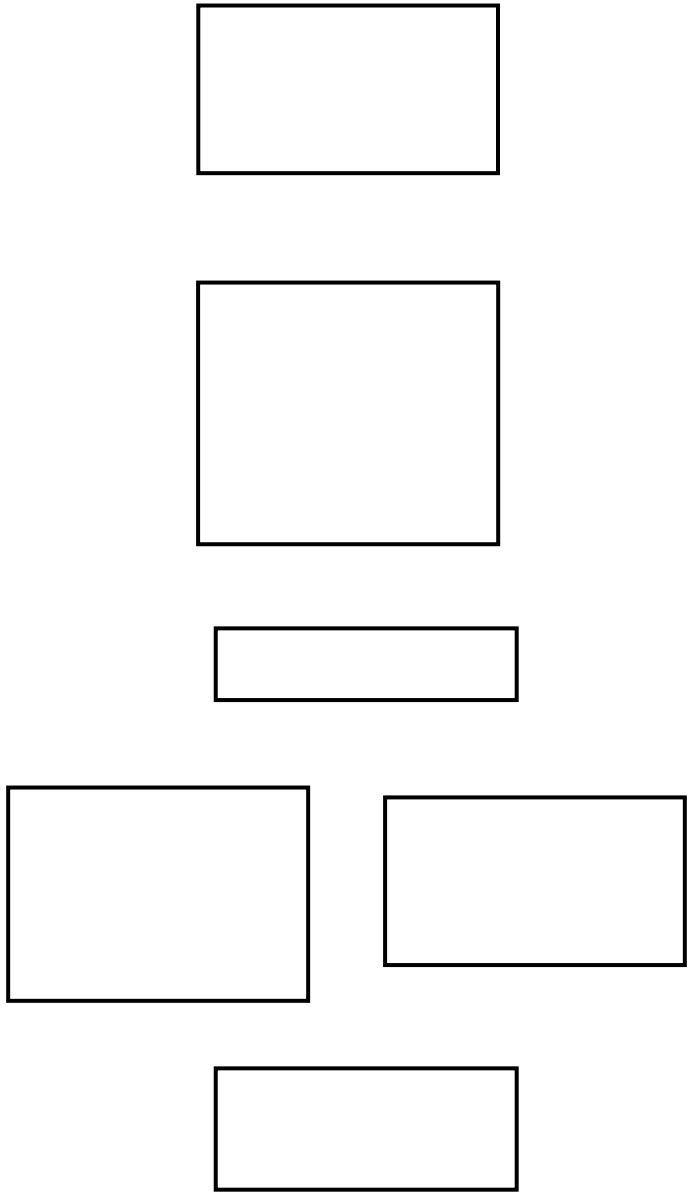


Code Generation

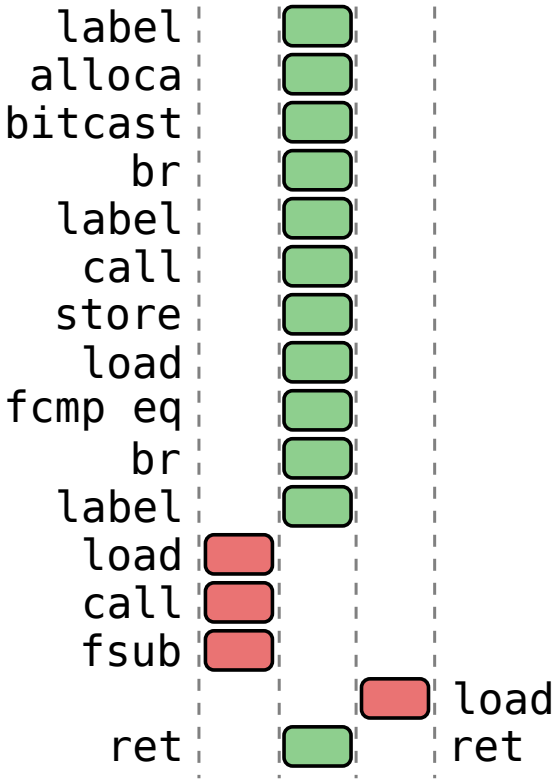
Our Technique



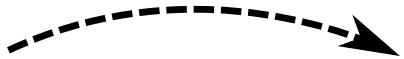
Code Generation

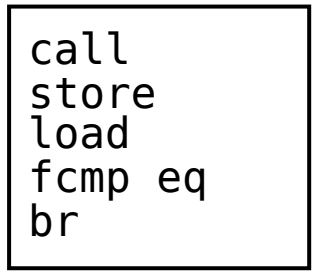


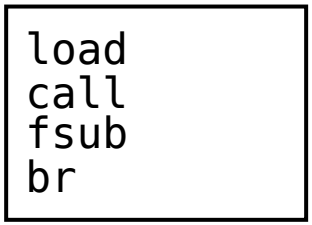
Our Technique



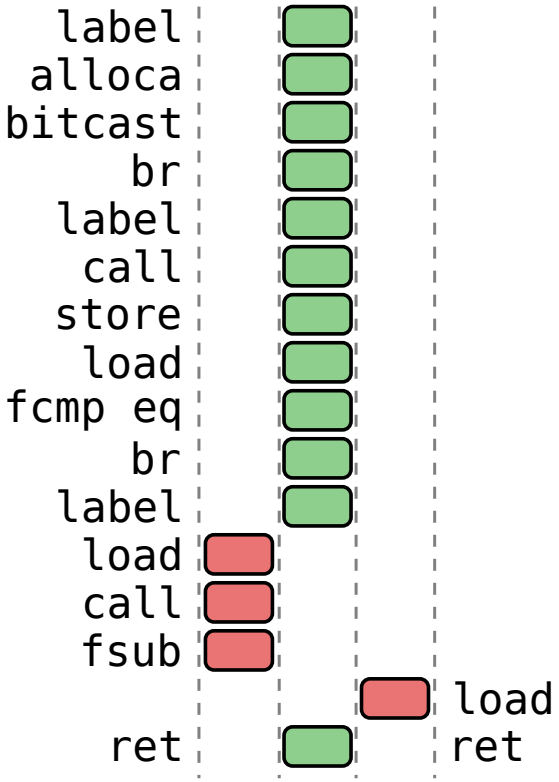
Code Generation



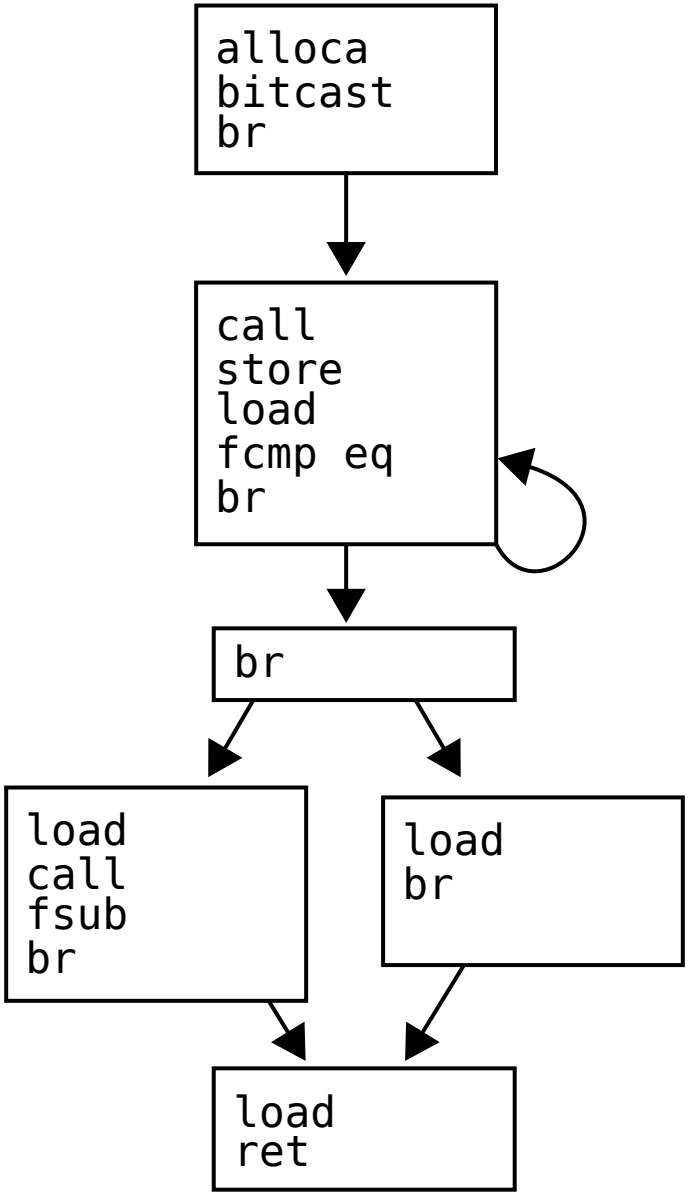




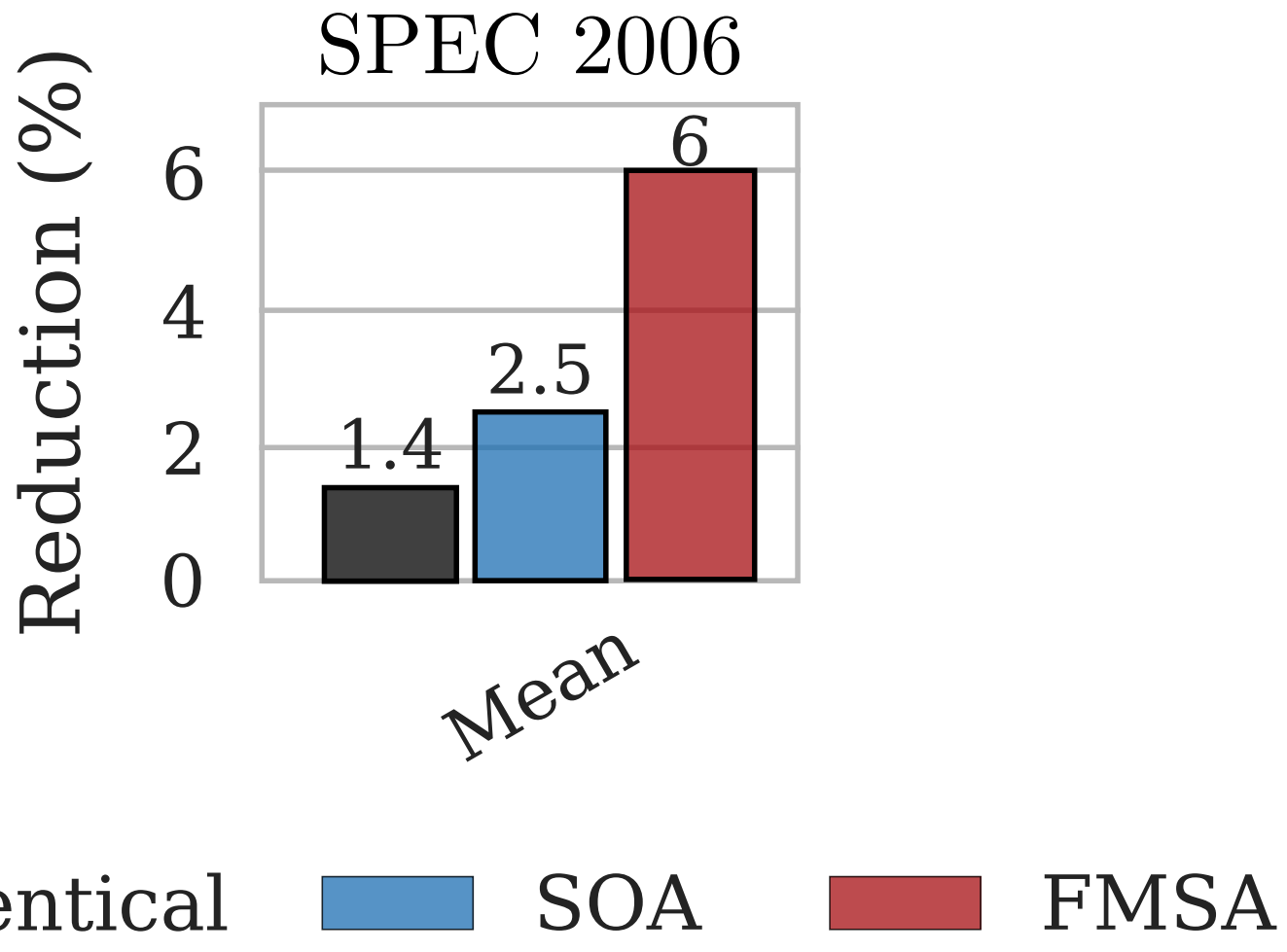
Our Technique



Code Generation



Evaluation



Evaluation