



# *CS415 Compilers*

## *Procedure Abstraction Part 2*

These slides are based on slides copyrighted by  
Keith Cooper, Ken Kennedy & Linda Torczon at Rice  
University

## Roadmap for the remainder of the course

- Project #3 - Local Dead-Code Elimination  
New due date: Wednesday May 4
- Homework #6 has been posted.  
New deadline: Friday, April 29
- Final exam on May 10 , 1:00pm (60 minutes in class)
- Grading Scheme
  - Exams:  $2 \times 30\%$  ( best two exams count )
  - Projects:  $3 \times 10\%$
  - Homeworks:  $5 \times 2\%$  ( best five homeworks count )

## EaC: Chapter 6.1 - 6.5

- Control Abstraction
  - Well defined entries & exits
  - Mechanism to return control to caller
  - Some notion of parameterization (usually)
- Clean Name Space
  - Clean slate for writing locally visible names
  - Local names may obscure identical, non-local names
  - Local names cannot be seen outside
- External Interface
  - Access is by procedure name & parameters
  - Clear protection for both caller & callee
- Procedures permit a critical separation of concerns

### Automatic & Local

- Keep them in the procedure activation record or in a register
- Automatic  $\Rightarrow$  lifetime matches procedure's lifetime

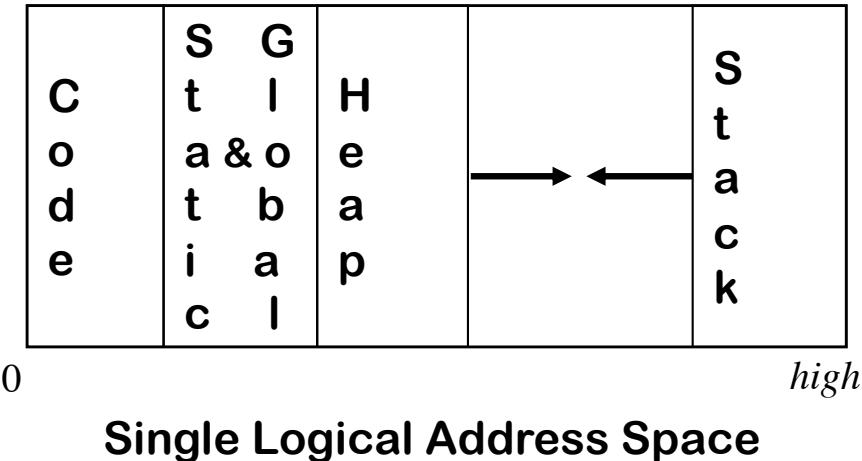
### Static

- Procedure scope  $\Rightarrow$  storage area affixed with procedure name
- File scope  $\Rightarrow$  storage area affixed with file name
- Lifetime is entire execution

### Global

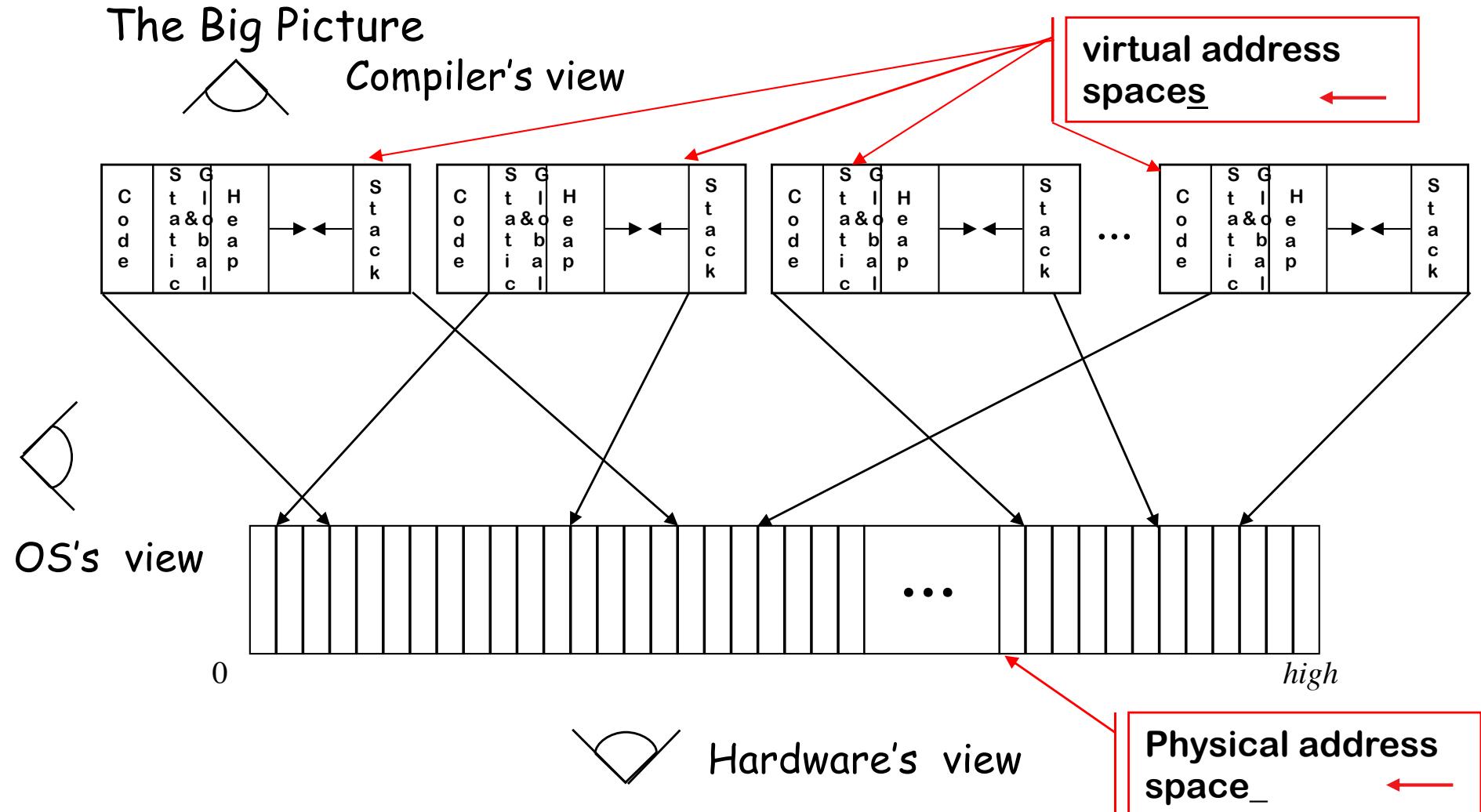
- One or more named global data areas
- One per program, ...
- Lifetime is entire execution

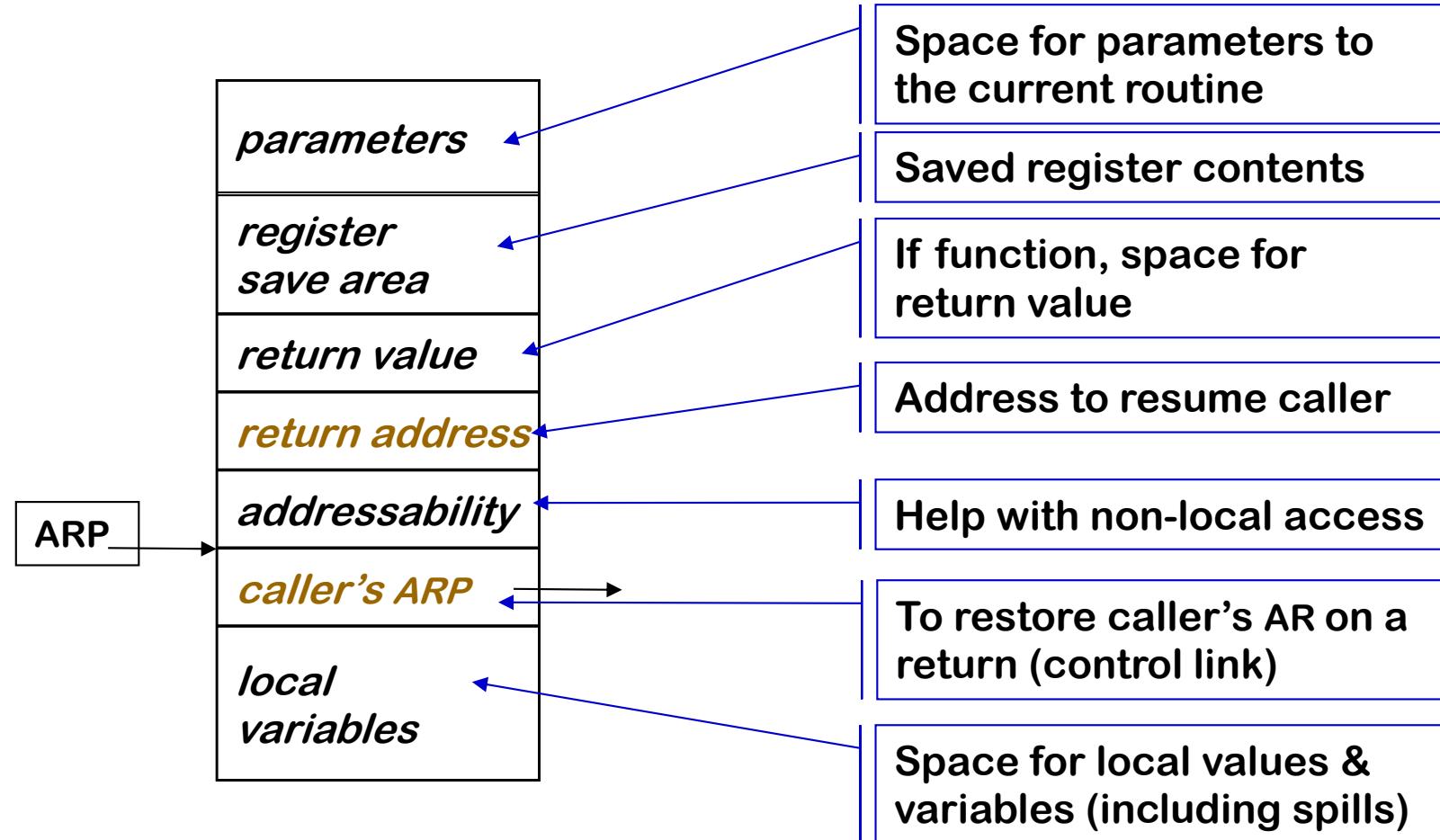
## Classic Organization



- Better utilization if stack & heap grow toward each other
- Very old result (Knuth)
- Code & data separate or interleaved

- Code, static, & global data have known size
- Heap & stack both grow & shrink over time
- This is a virtual address space





One AR for each invocation of a procedure

0

```
int r (...) { // declaration
    int d, s;
```

```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

}

0

```
int r (...) { // declaration
    int d, s;
```

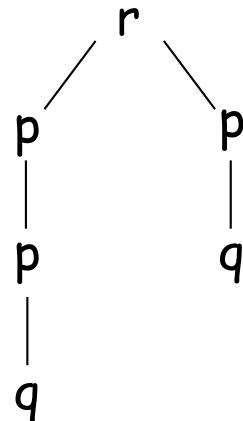
```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

}

cs415, spring 22

(1) dynamic  
activation tree



Lecture 24

9

0

```
int r (...) { // declaration
    int d, s;
```

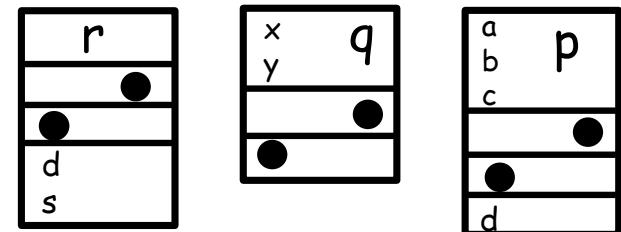
```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

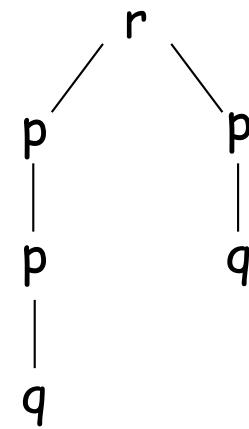
}

cs415, spring 22

(2) dynamic activation records in runtime stack



(1) dynamic activation tree



0

```
int r (...) { // declaration
    int d, s;
```

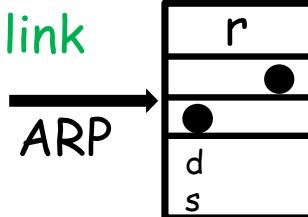
```
1 int q (x,y) // declaration
      int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
      int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

}

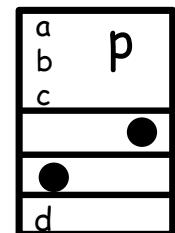
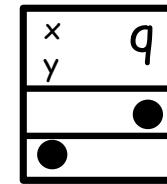
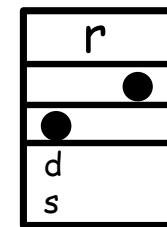
cs415, spring 22

control link



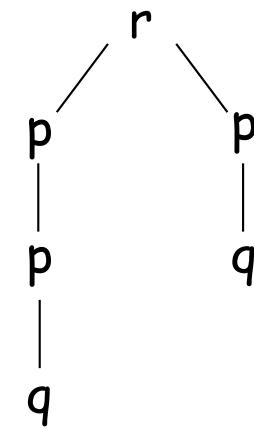
access link

(2) dynamic activation records in runtime stack



(1) dynamic activation tree

RUNTIME STACK



Lecture 24

11

0

```
int r (...) { // declaration
    int d, s;
```

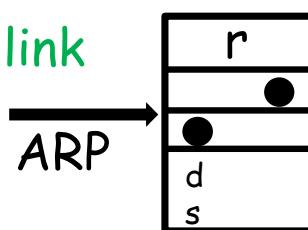
```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

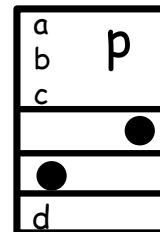
}

cs415, spring 22

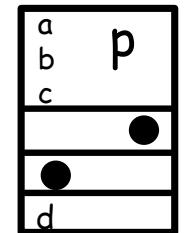
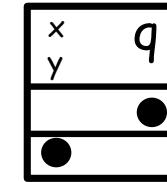
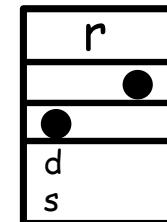
control link



access link

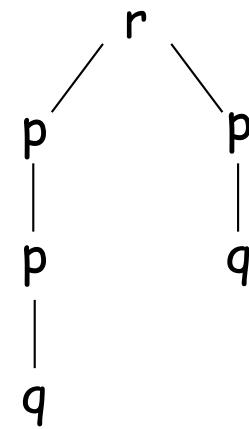


(2) dynamic activation records in runtime stack



(1) dynamic activation tree

RUNTIME STACK



Lecture 24

12

0

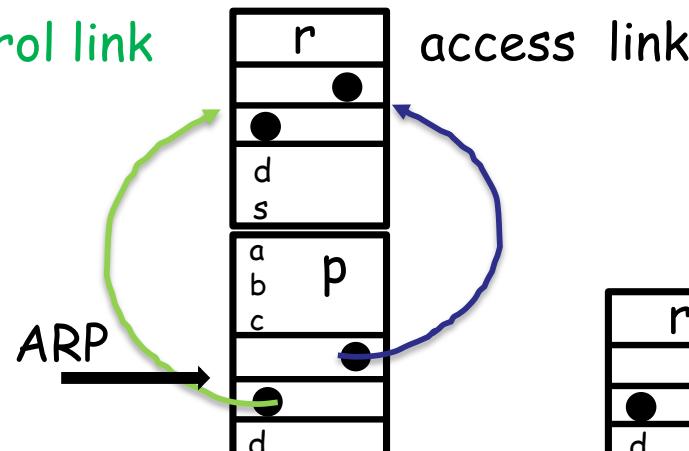
```
int r (...) { // declaration
    int d, s;
```

```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

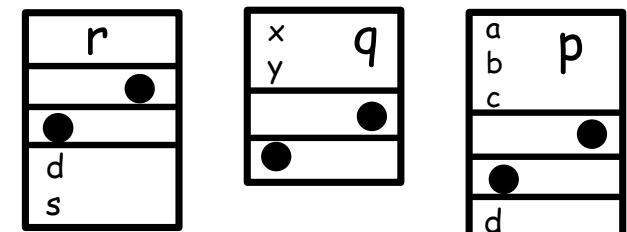
```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

} cs415, spring 22

control link

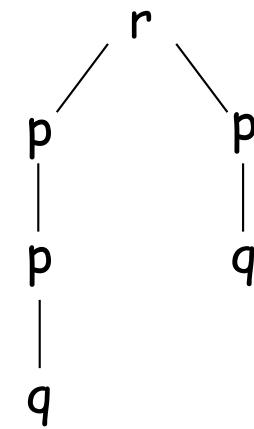


(2) dynamic activation records in runtime stack



(1) dynamic activation tree

RUNTIME STACK



0

```
int r (...) { // declaration
    int d, s;
```

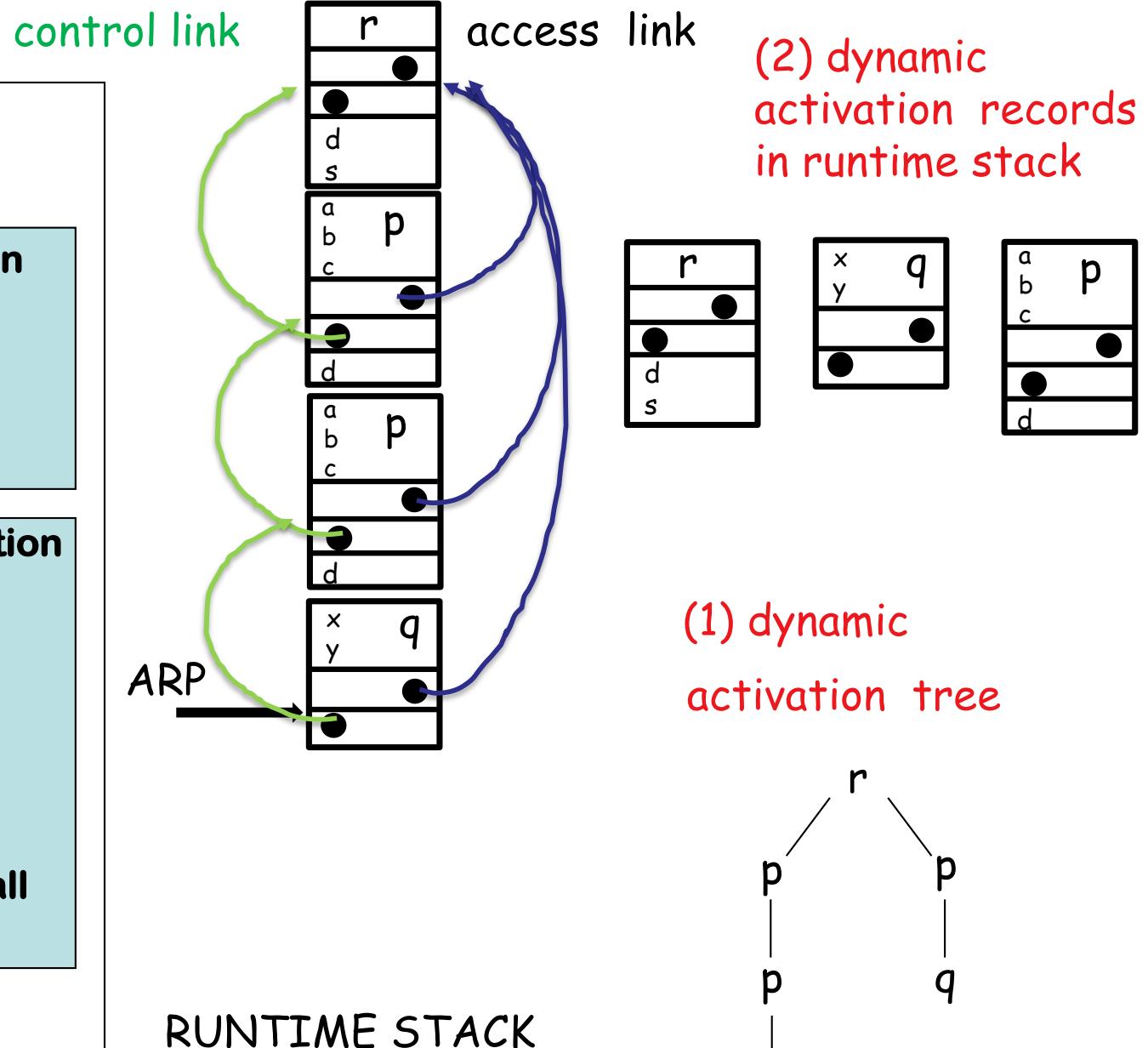
```
1 int q (x,y) // declaration
    int x,y;
{
    return x + y + d;
}
```

```
1 int p (a,b,c) // declaration
    int a, b, c;
{
    int d;
    if (...)
        d = q (c,b); // call
    else
        d = p (a, d, c); // call
}
s = p(10, d, s); // call
s = p(11, s, d); // call
```

}

cs415, spring 22

control link



(2) dynamic activation records in runtime stack

Work on the project!

More on procedure abstraction

Wrap-up parsing: SLR(1) and LALR(1)

Read EaC: Chapter 3.4