Alt Programları Uygulamak

- Düzgün İfade ..... L(R) dilinde örnek katarlar
  - rakam [0-9] .... “0”, “1”, “2”, ...
  - poztamsayı = rakam^{+} ..... “8”, “412”, ...
  - tamsayı = (-| ε) poztamsayı ..... “-23”, “34”, ...
  - reelsayı = tamsayı(ε |(.poztamsayı))“-1.56”, “12”, “1.056”
    (dikkat: bu tanım “.58” ve “45.” sözcüklerine izin vermez)
  - harf [a-z] ..... “a”, “b”, “c”,....
  - değişken_adı = harf(harf | rakam)*..... “toplam”, “sayac”,...

Özellik Gramerleri: Örnek

- Sentaks

  <assign> -> <var> = <expr>

  <expr> -> <var> + <var> | <var>

  <var> A | B | C

- actual_type: <var> ve <expr> ile sentezlenmiştir
- expected_type: <expr> ile miras bırakılmıştır

Özellik Gramerleri: Örnek

- Sentaks kuralı: **<expr> → <var>[1] + <var>[2]**

  Semantik kurallar:

<expr>.actual_type ← <var>[1].actual_type

  Karşılaştırma belirtimi (Predicate):

<var>[1].actual_type == <var>[2].actual_type

<expr>.expected_type == <expr>.actual_type

- Sentaks kuralı: **<var> → id**

  Semantik kuralı:

<var>.actual_type ← lookup (<var>.string)

Özellik Gramerleri – Bir Örnek

- Nitelikler: *actual_type* (sentezlenen nitelik), *expected_type* (miras kalan nitelik)

1. Sentaks kuralı: <assign> → <var> = <expr>
   Semantik kuralı: <expr>.*expected_type* ← <var>.*actual_type*
2. Sentaks kuralı: <expr> → <var>[2] + <var>[3]
   Semantik kuralı: <expr>.*actual_type* ←
   if (<var>[2].*actual_type* = int) and
   (<var>[3].*actual_type* = int) then int
   else real
   endif
   Predicate: <expr>.*actual_type* = <expr>.*expected_type*
3. Sentaks kuralı: <expr> → <var>
   Semantik kuralı: <expr>.*actual_type* ← <var>.*actual_type*
   Predicate: <expr>.*actual_type* = <expr>.*expected_type*
4. Sentaks kuralı: <var> → A | B | C
   Semantik kuralı: <var>.*actual_type* ← look-up(<var>.*string*)

Nitelik Değerlerini Hesaplama – Nitelikleri Değerlendirme

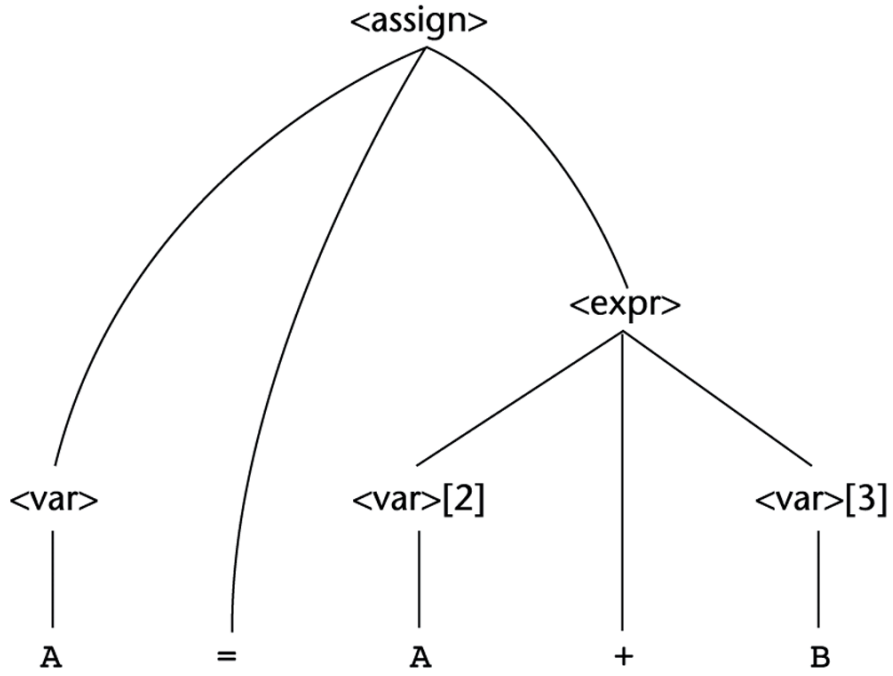
Cümle: A = A + B

1. <var>.*actual_type* ← look-up(A) (Kural 4)
2. <expr>.*expected_type* ← <var>.*actual_type* (Kural 1)
3. <var>[2].*actual_type* ← look-up(A) (Kural 4)
   <var>[3].*actual_type* ← look-up(B) (Kural 4)
4. <expr>.*actual_type* ←
   int ya da real (Kural 2)
5. <expr>.*expected_type* =
   <expr>.*actual_type*

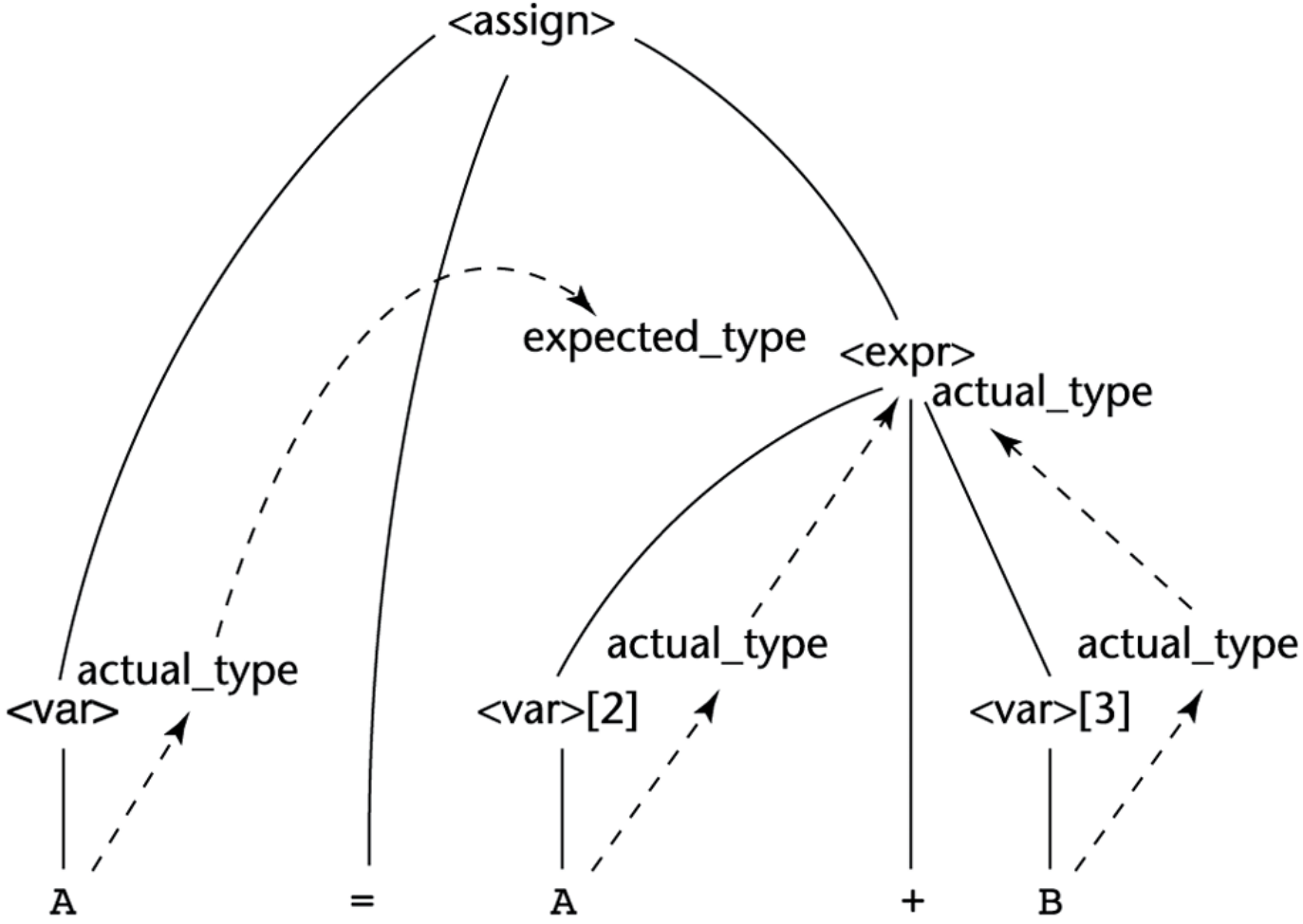
TRUE ya da FALSE'tur (Kural 2)

Gramer:

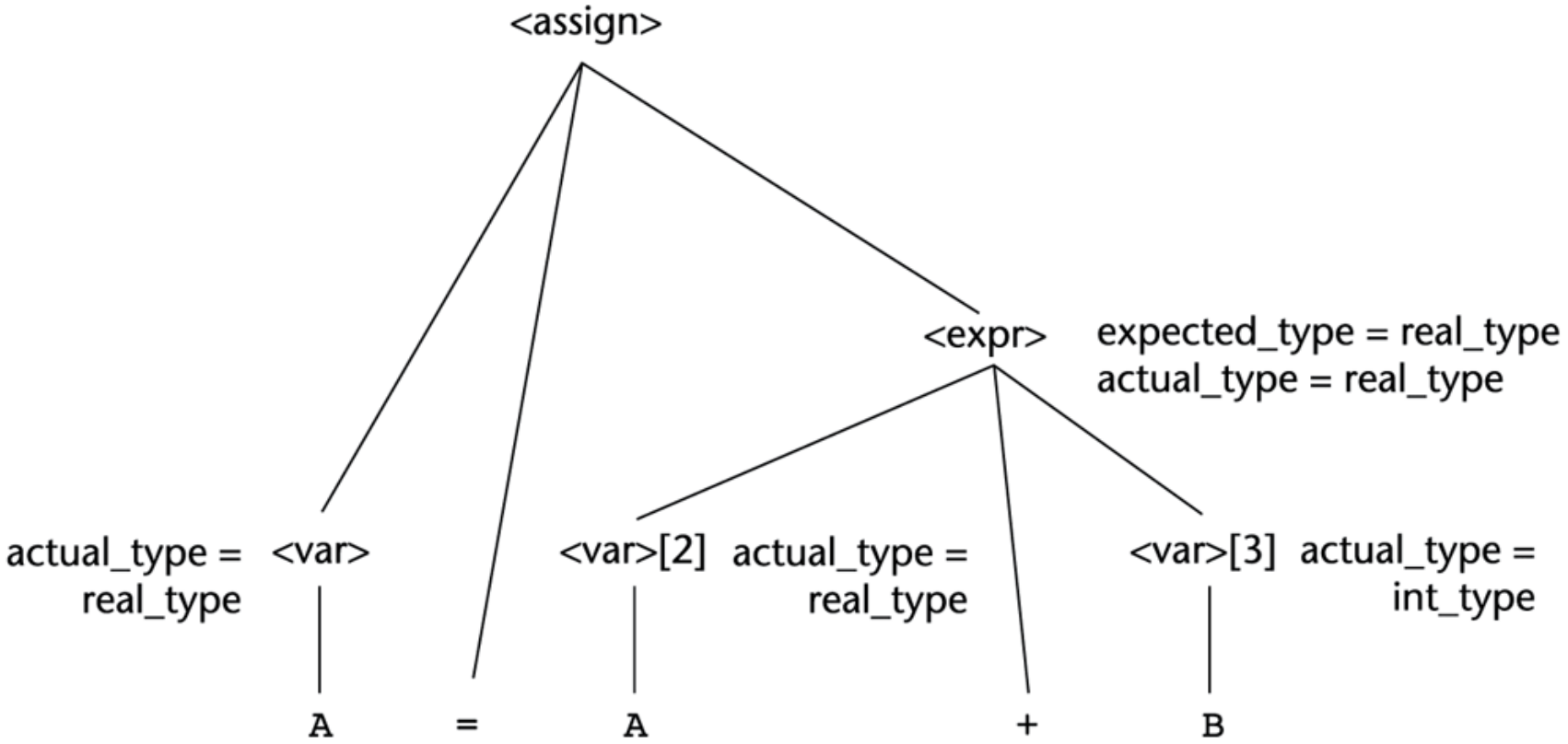
<assign> → <var> = <expr>
<expr> → <var> + <var> | <var>
<var> → A | B | C



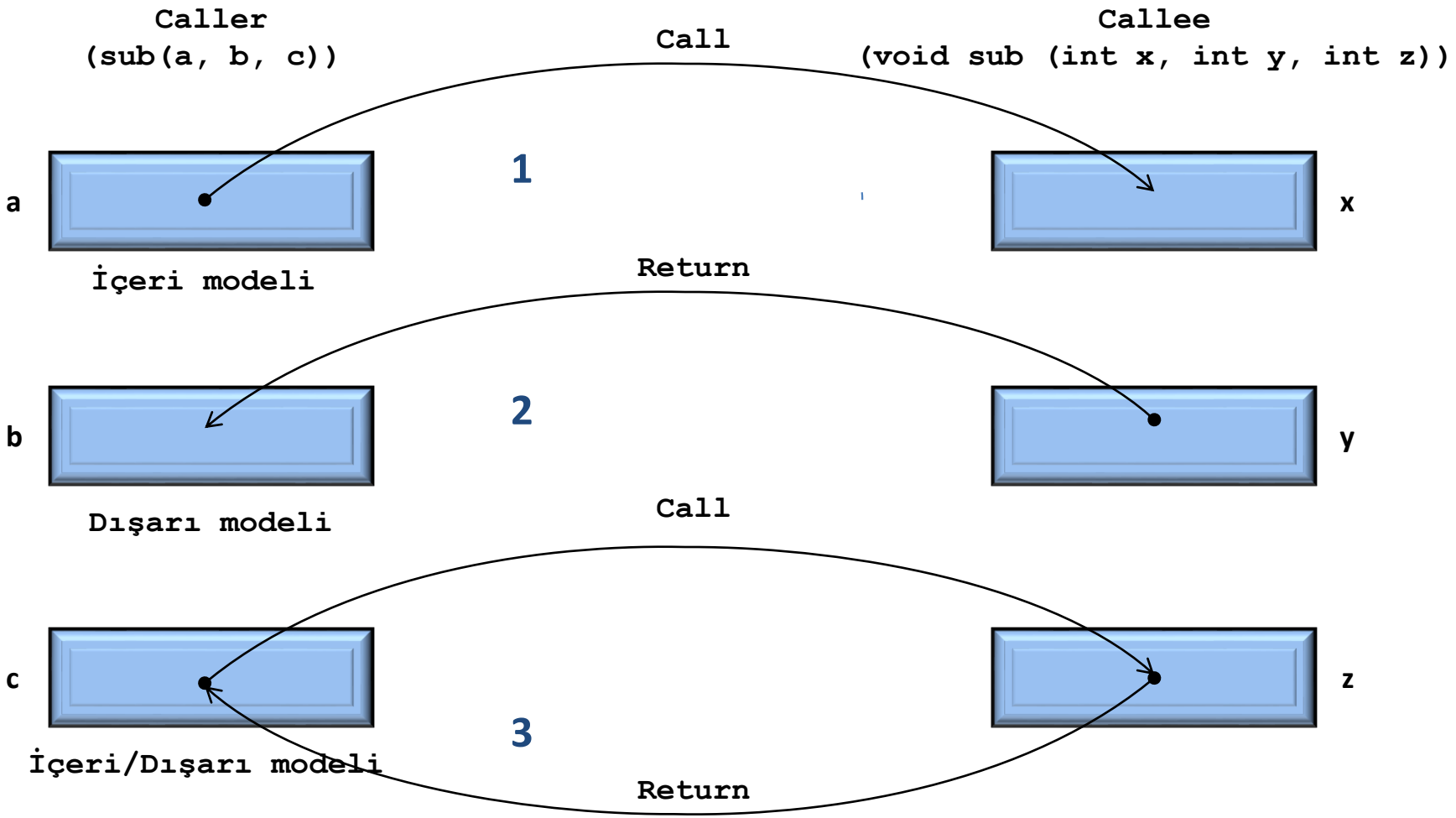
Nitelik Değerlerini Hesaplama –
Ayrıştırma Ağacında Nitelik Akışı



Nitelik Değerlerini Hesaplama –
Tam bağlanmış nitelik ağacı



Parametre Aktarım Yöntemleri



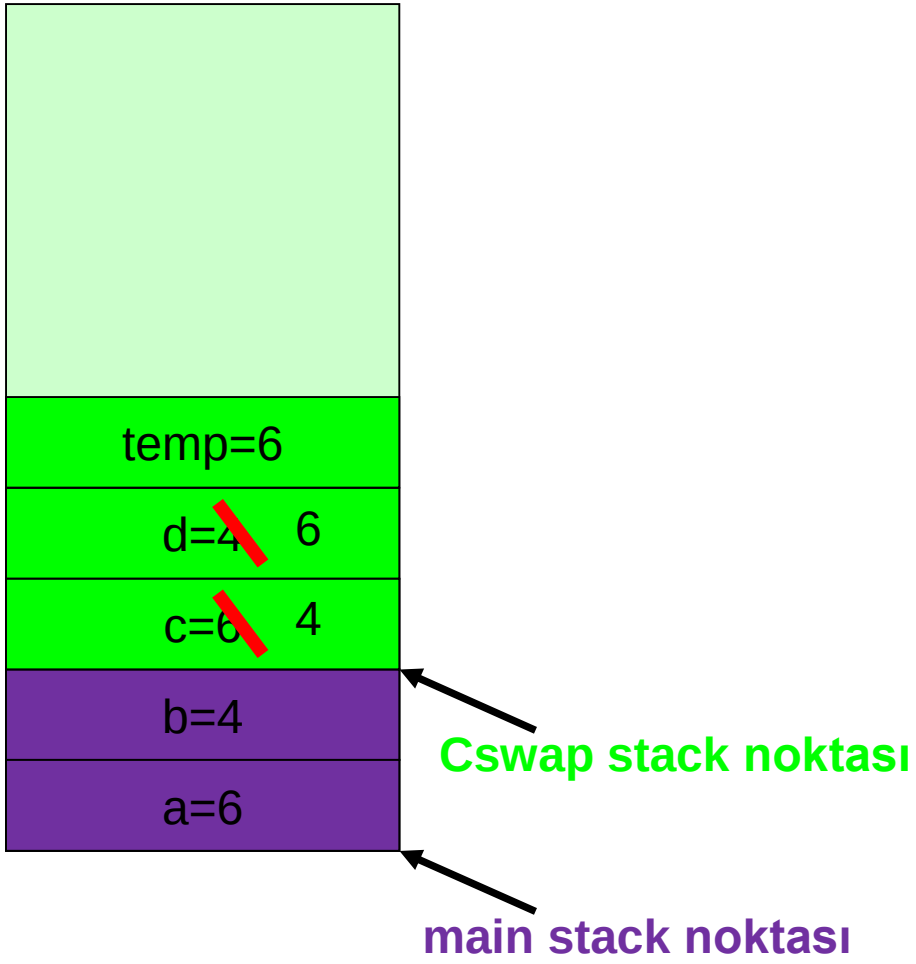
Parametre Aktarım Yöntemleri

- Resmi parametreler ve gerçek parametreler arasındaki veri akışı, parametre aktarım yöntemlerine göre gerçekleştirilir.
- Değer ile çağırma, sonuç ile çağırma, değer ve sonuç ile çağırma yöntemlerinde, gerçek ve resmi parametreler arasındaki veri fiziksel olarak kopyalanarak aktarılmakta, başvuru ile çağırma yönteminde ise veri yerine verinin erişim yolu aktarılmaktadır.

Değer ile Çağırma (*Örnek*)


```
main( ) {
   int  a = 6;
   int  b = 4;
   Cswap(a, b);
   //  a = 6
   //  b = 4
}
```

```
Cswap(int c, int d) {
   int  temp = c;
   c = d;
   d = temp;
}
```



Değer ile Çağırma (*Call by Value*)


```
void f(int x) {
  x = 3;
}

int main() {
  int x = 0;
  f(x);
  printf("%d\n", x);
}
```

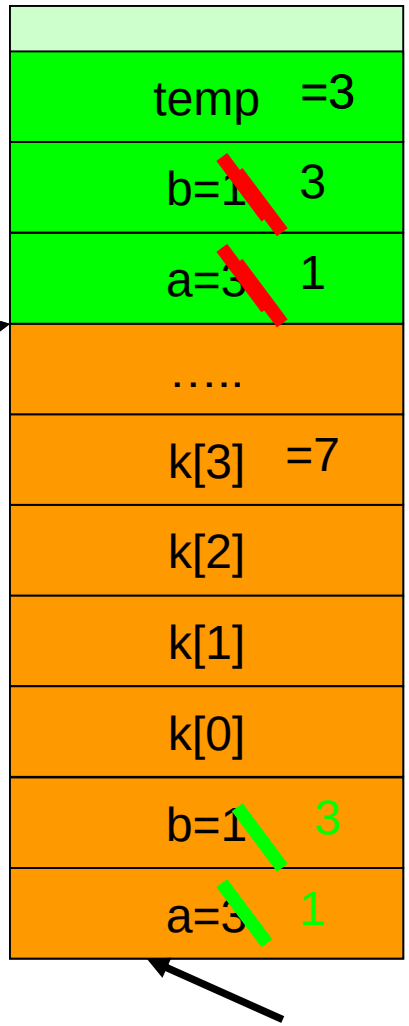
Sadece gerçek parametreden formal parametreye değer geçişi olduğu için en güvenilir parametre aktarım yöntemidir

Programın çıktısı= 0 olacaktır.

Değer ve Sonuç ile Çağırma (*Örnek*)


```
integer  a = 3 ;
integer  b = 1 ;
integer  k[10] ;
k[3] = 7;
swap(a, b);
```

swap stack noktası



main stack noktası

```
procedure swap(a : in out integer,
               b : in out integer) is
    temp : integer;
    begin
        temp :=  a ;
        a := b ;
        b := temp ;
    end swap;
```

```
int x=0;
int main()
 {
   f(x);
……
}
void f(int a) {
   x=3;
   a++;
}
```

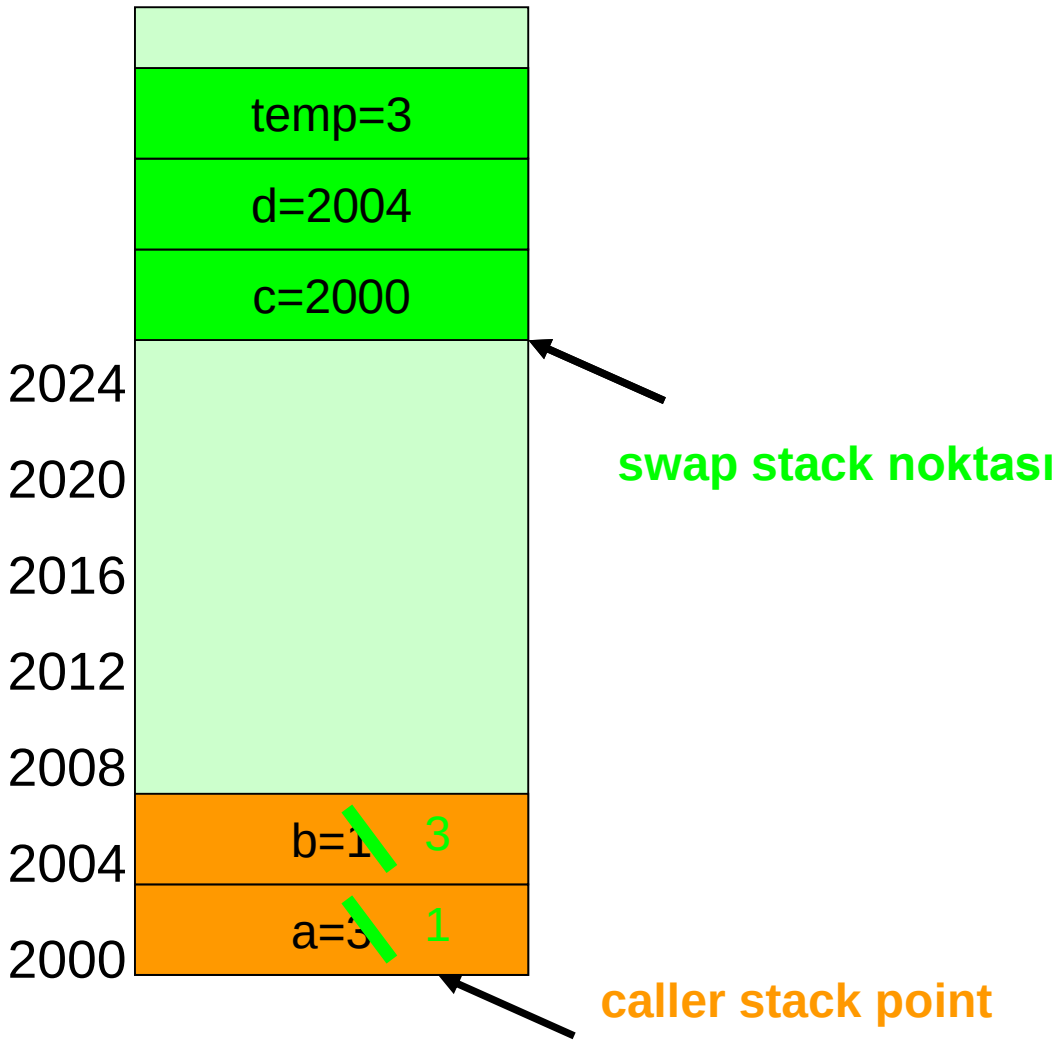
x'in son değeri değer-sonuç aktarımına göre 1 olacaktır.

Parametreler için birden çok bellek yeri gerekmesi ve değer kopyalama işlemlerinin zaman almaktadır.

Başvuru ile Çağırma (*Örnek*)


```
caller( ) {
   int  a = 3 ;
   int  b = 1 ;
swap(&a, &b) ;
}
```

```
swap(int *c, int *d ) {
   temp = *c;
   *c = *d ;
   *d = temp ;
}
```



```
x : integer;
  procedure foo(y : out integer)
  y := 3;
  print x;
  . . .
x := 2;
foo(x);
print x;
```

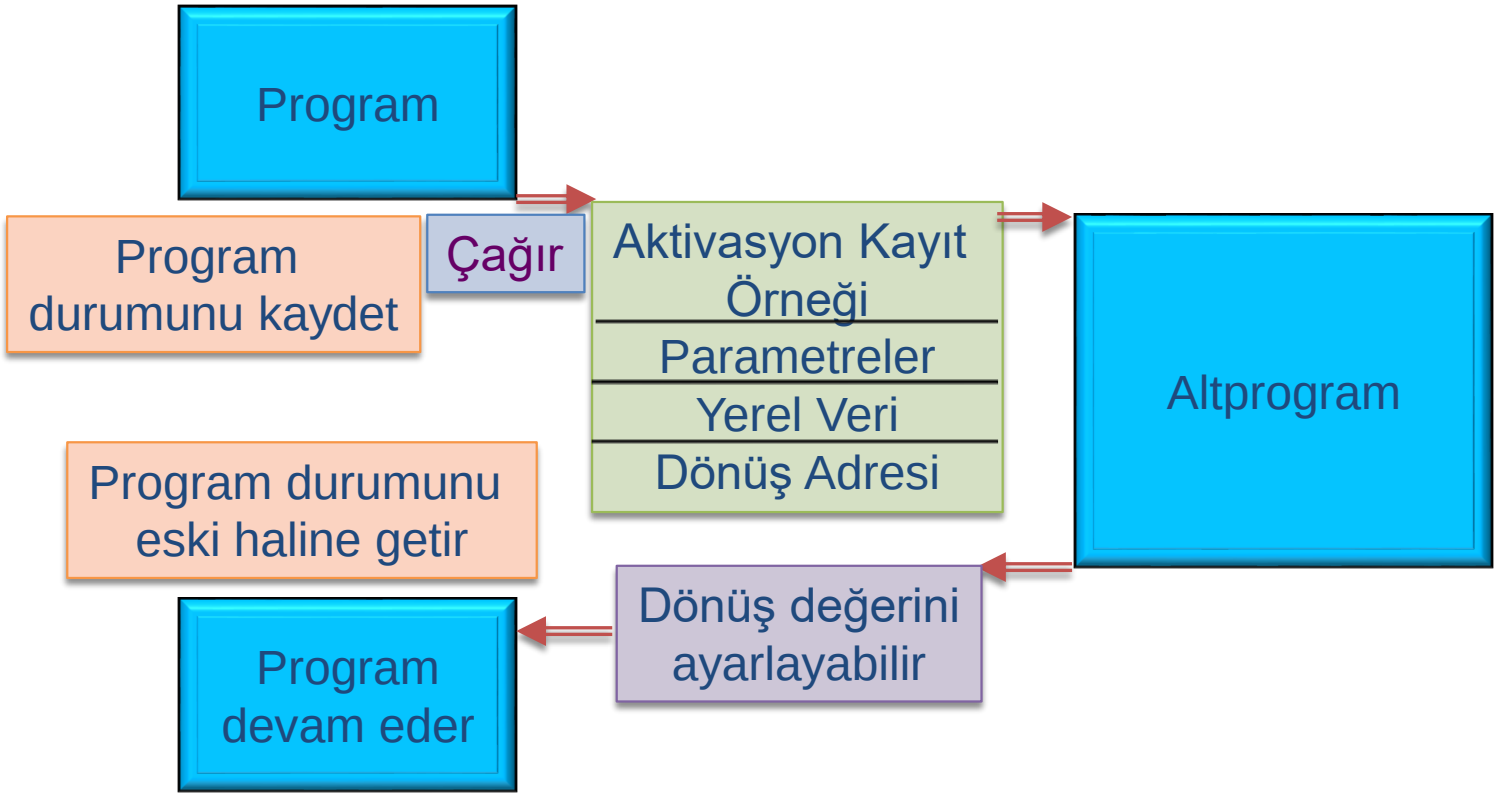
Eğer y başvuru ile geçirilmişse program iki kere 3 yazar.

Eğer y değer/sonuç ile geçirilmişse önce 2 sonra 3 yazar.

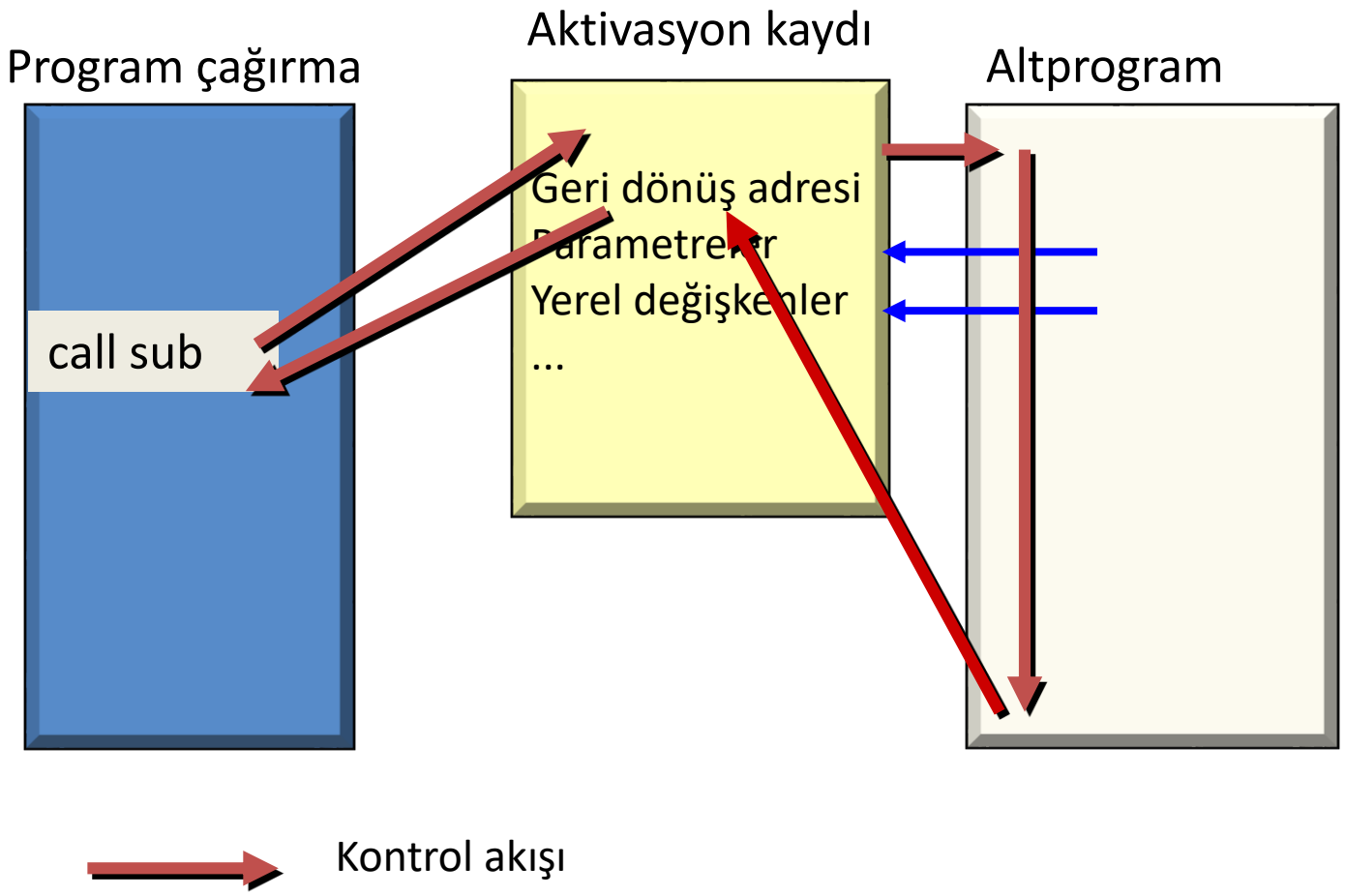
Çağrıların ve döndükleri değerlerin genel anlam analizi

- Bir dilde altprogramların çağrılması ve dönmesi işlemlerine altprogram bağlanması (subprogram linkage) denir.
- Parametrelerin alt programa nasıl geçileceği belirlenmelidir.
- Çağıranın değerleri korunmalıdır.
- Dönüşte çağıranın doğru noktasına, doğru değerlerle dönülmesi temin edilmelidir.
- İç içe altprogramlar varsa, yerel olmayan değişkenlere erişim sağlanmalıdır.

Genel Tasarım



Genel Tasarım



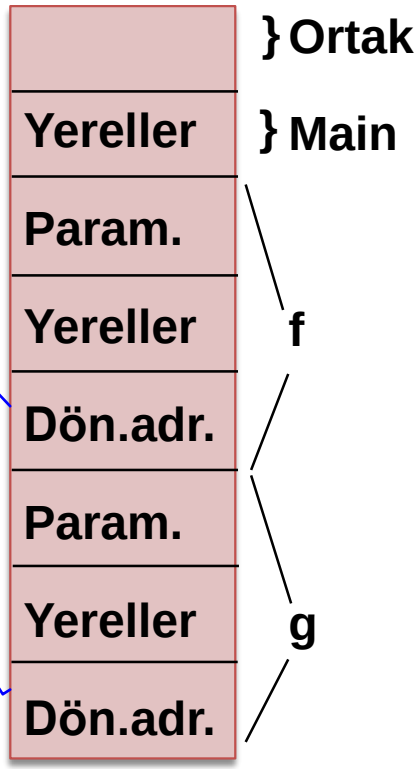
"Basit" altprogramların gerçekleştirilmesi

- Basit altprogramdan kastımız, bütün yerel değişkenleri statik olan ve iç içe çağrılamayan altprogramlar (örn. Fortran I).
- Çağrı anlam analizi:
  1. Çağıran programın çalışma durumunu sakla.
  2. Parametre geçme işlemlerini yap.
  3. Dönüş adresini çağrılana geç
  4. Kontrolü çağrılana ver.

Fortran aktivasyon bilgisi

- Tüm hafızaya statik olarak yerleştirebilir

```
main
   ...
   f (...)
   ...
proc f (...)
   ...
   g (...)
   ...
proc g( )
   ...
   g (...)    -- !!!
```



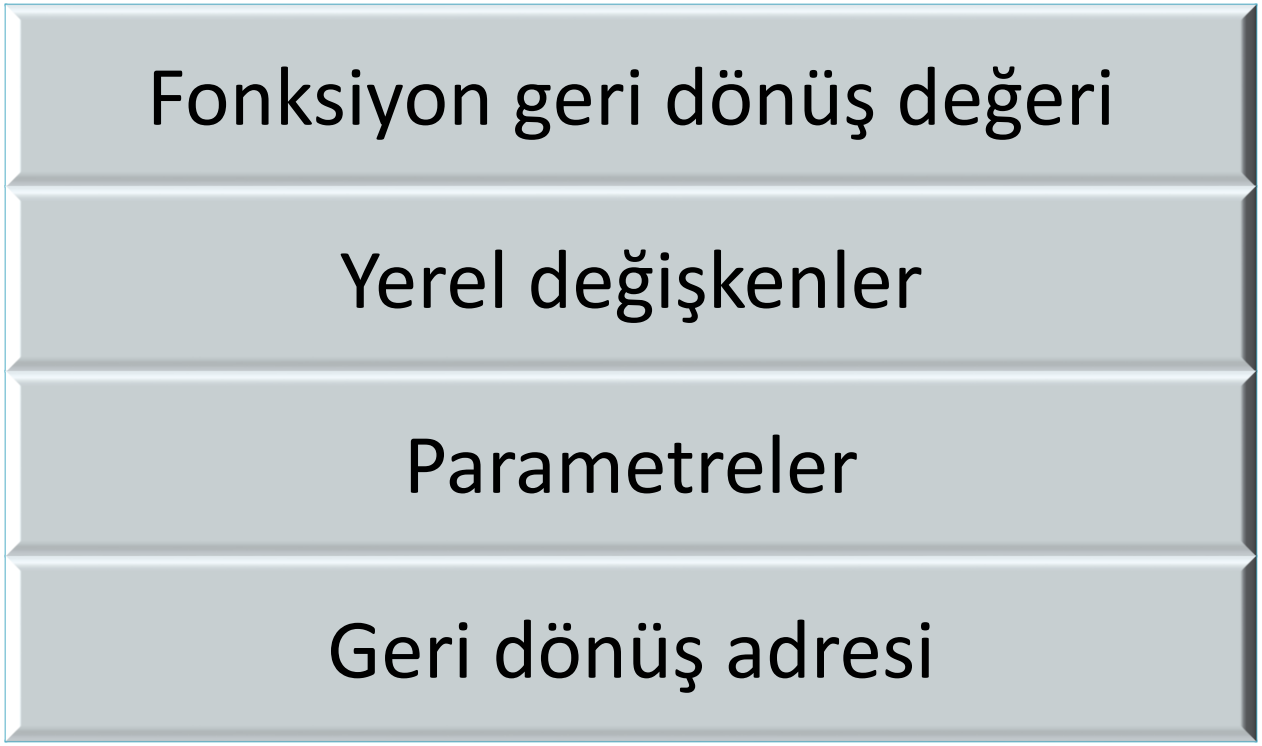
"Basit" altprogramların gerçekleştirilmesi

- Dönüş Analizi (Return Semantics):
  - 1. Eğer değer-sonuç ile geç (pass-by-value-result) parametreler kullanılmışsa değerlerini gerçek parametrelere geçir.
  - 2. Eğer dönen fonksiyonsa, döneceği değeri çağıranın bulması gereken yere yerleştir.
  - 3. Çağıranın çalışma ortamını yeniden yapılandır.
  - 4. Kontrolü çağırana geri ver.

"Basit" altprogramların gerçekleştirilmesi

- Gerekli depolama: Çağıran durum bilgileri, parametreleri, dönüş adresi ve döneceği değer (eğer fonksiyonsa).
- Altprogramın çalışan kod olmayan, altprogram verilerinin tutulduğu kısmının biçimi veya yerleşim planına etkinleştirme kaydı (activation record) denir. Bu kaydın sonraki sayfalarda göreceğimiz gibi belli bir formatı olur.
- Altprogram çağrıldığı zaman belli değerlerle doldurulmuş haline somutlaşan gerçekleşme kaydı denir (instance of activation record).

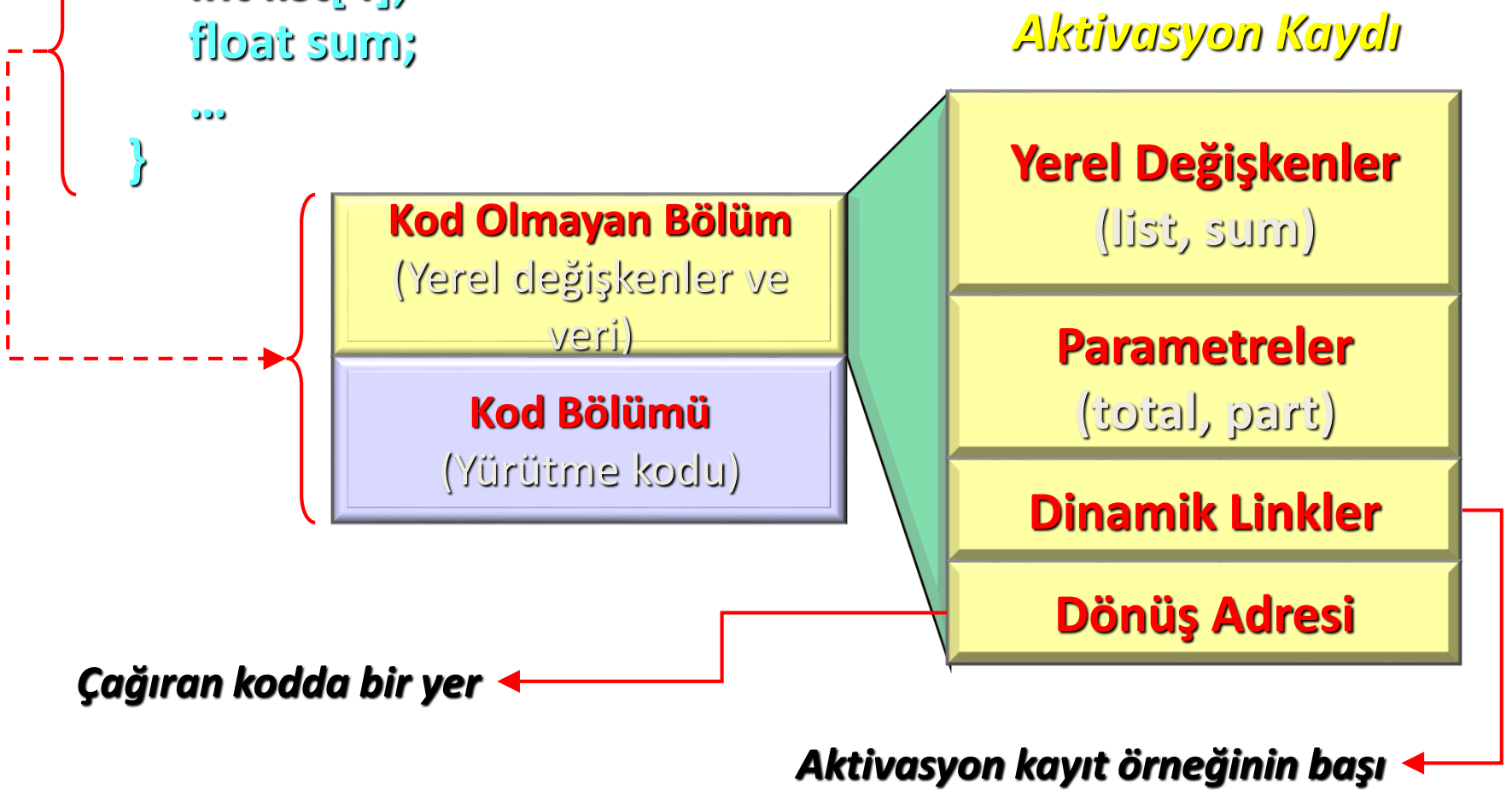
"Basit" altprogramların etkinleştirme kaydı

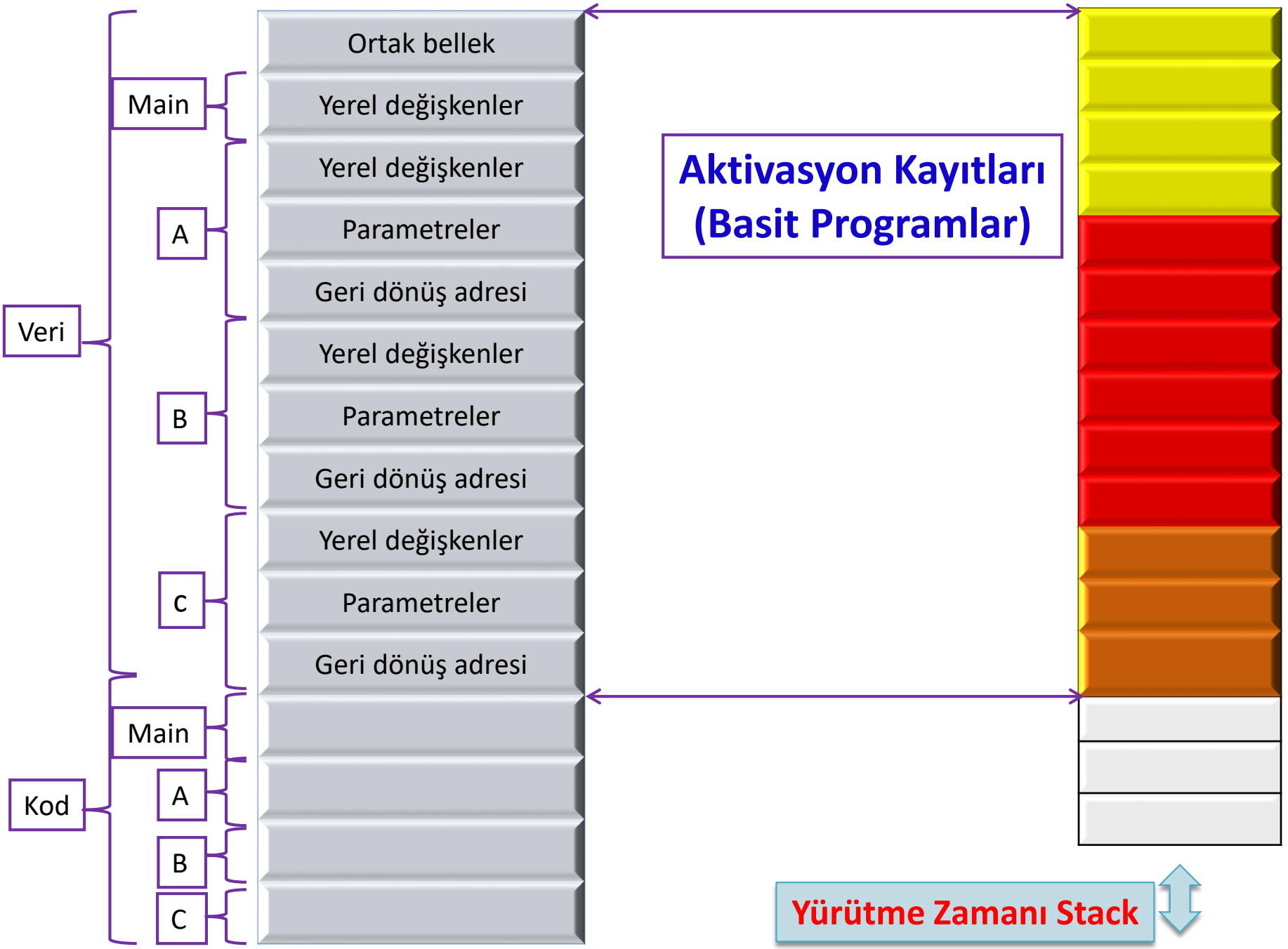


"Basit“ altprogramların etkinleştirme kaydı ve kodu

- Etkinleştirme kaydı verileri statik bellekte statik olarak saklanır.
- Bu nedenle somutlaşmış tek etkinleştirme kaydı olabilir.
- Bu nedenle kullanım olarak kolay, erişim daha hızlı, ancak özyinelemeyi desteklemez.
- İlk Fortran derleyicileri bu tip etkinleştirme kaydı tutuyordu.

```
void sub (float total, int part) {
    int list[4];
    float sum;
    …
}
```

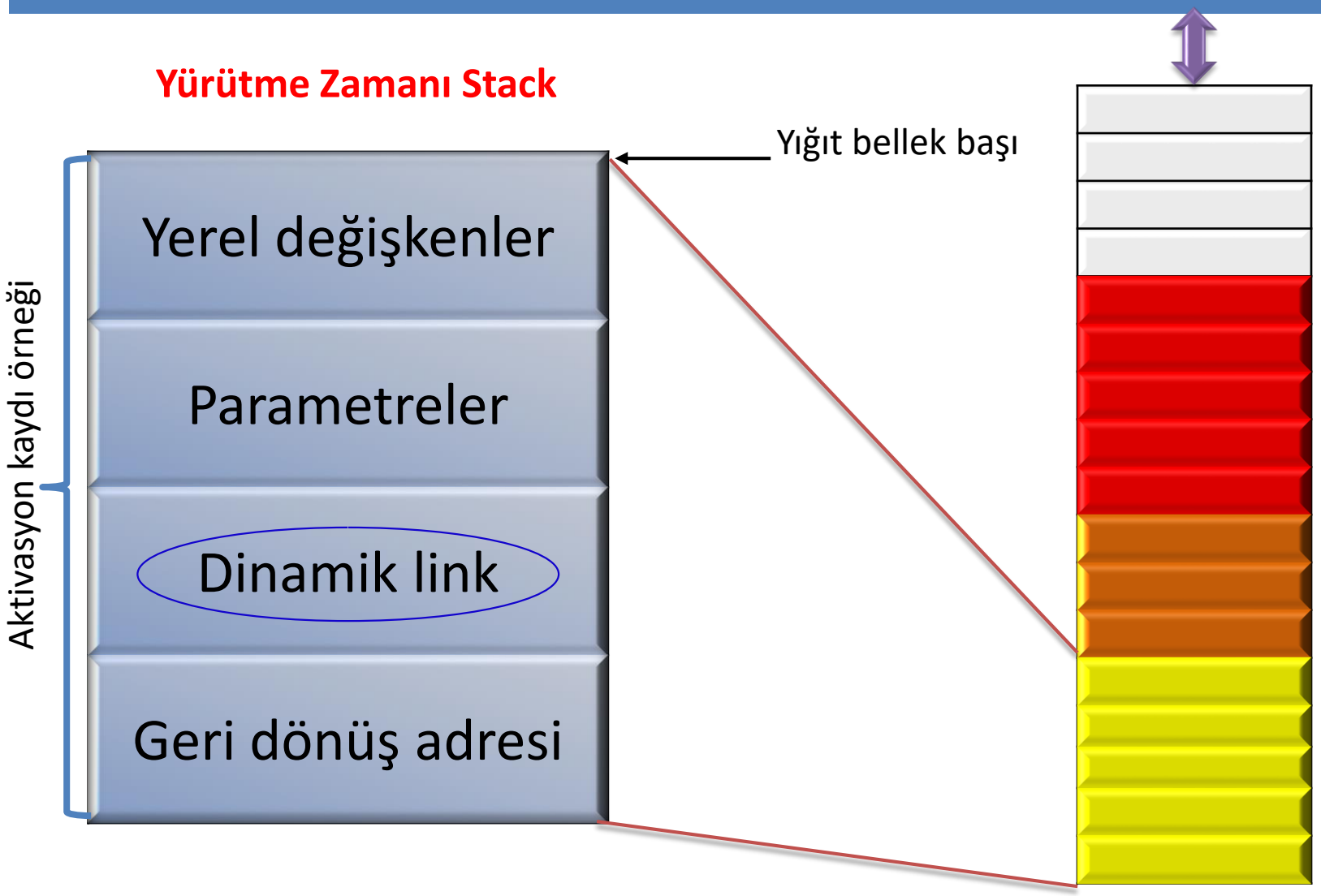



Ortak bellek
Main
Yerel değişkenler
Yerel değişkenler
A
Parametreler
Geri dönüş adresi
Veri
Yerel değişkenler
B
Parametreler
Geri dönüş adresi
Yerel değişkenler
C
Parametreler
Geri dönüş adresi
Main
A
Kod
B
C
Aktivasyon Kayıtları
(Basit Programlar)
Yürütme Zamanı Stack

Altprogramları yığıt dinamik (Stack-Dynamic) yerel değişkenlerle gerçekleştirmek

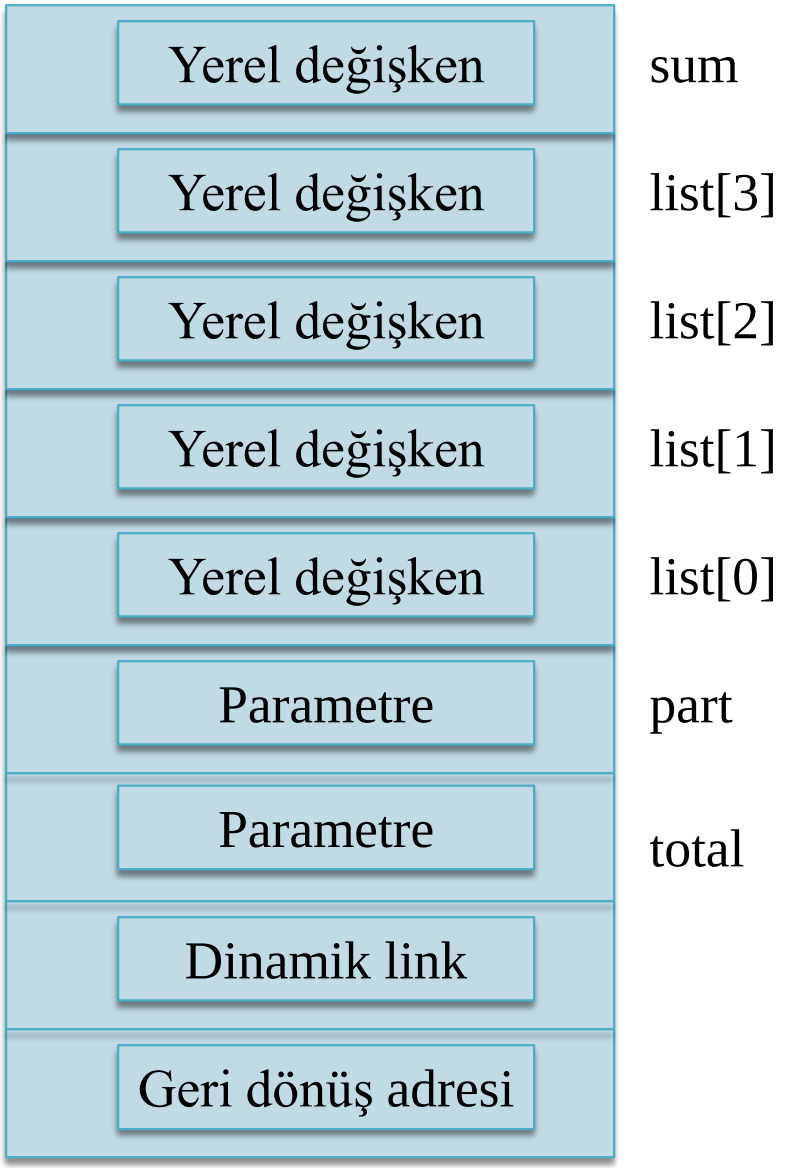
- Daha karmaşık çünkü:
  - Derleyici altprogram yerel değişkenlerine yığıt bellekte örtülü bellek tahsis ve geri alma işlemleri için kod hazırlamalıdır.
  - Özyineleme desteklenebildiğinden altprogramın aynı anda birden çok peş peşe çağırılabilmesini destekler, bu da birden çok etkinleştirme kodu somutlaşmasına neden olur.
  - Bazı dillerde yerel dizilimlerin (local arrays) boyu çağrı sırasında belirlendiğinden, etkinleştirme kodunun içinde yer alan bu tip değişkenler etkinleştirme kodunun boyunun dinamik olmasını gerektirir.

Tipik Altprogramları yığıt dinamik yerel değişkenlerle gerçekleştirme etkinleştirme kaydı



Tipik Altprogramları yığıt dinamik yerel değişkenlerle gerçekleştirme

```
void sub(float total, int part) {
    int list[4];
    float sum;
    …
}
```



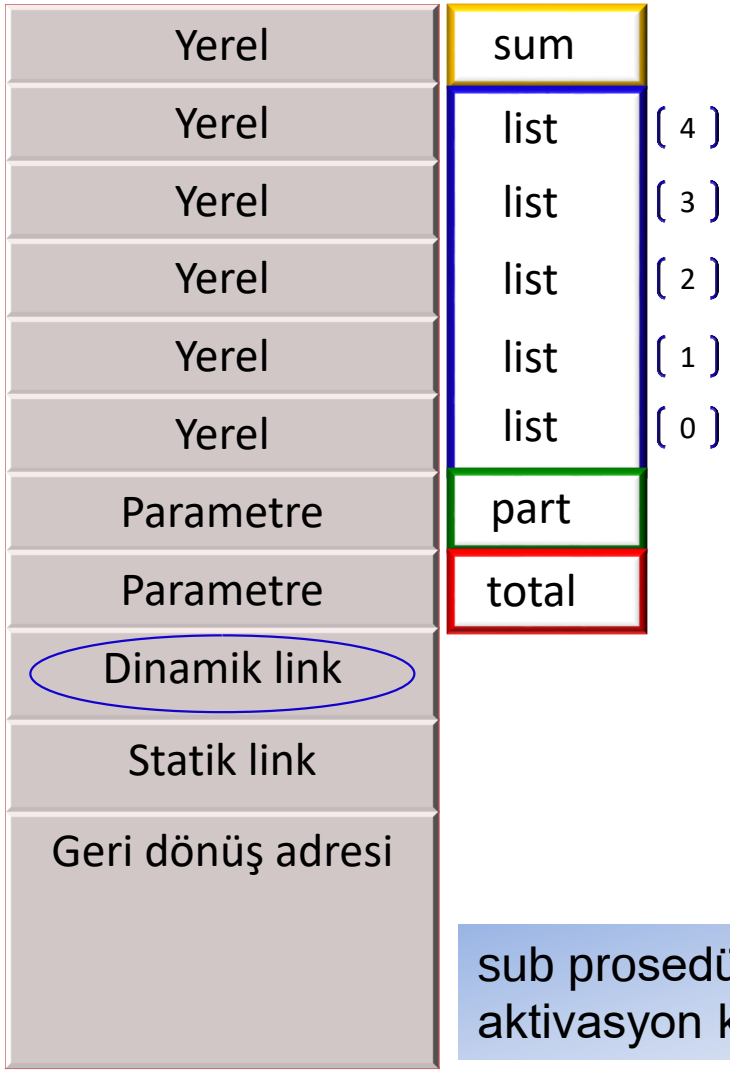
Altprogramları yığıt dinamik yerel değişkenlerle gerçekleştirmek

- Etkinleştirme kaydı formatı statik fakat boyutları dinamiktir.
- “Dinamik link" çağıran programın etkinleştirme kaydının başını gösterir.
- Etkinleştirme kaydı dinamik olarak altprogram çağrıldığında üretilir.

C fonksiyonu örneği


```
void sub (float total,
          int part) {
   int list[5];
   float sum;
   …
}
```

- statik link: yerel olmayan değişkenler referansı için kullanılır
- dinamik link: çağıranın AR'sinin başını gösterir



sub prosedürü için aktivasyon kaydı

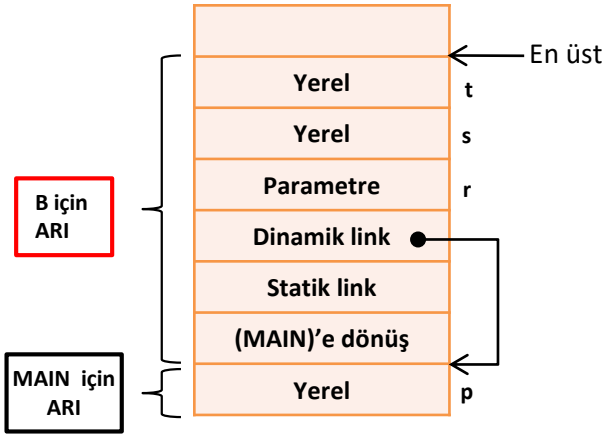
AR: Activation Record (Etkinleştirme (Aktivasyon) Kaydı)

Özyinelemesiz C programı örneği – Program için yığıt bellek içeriği


```
void A(int x) {
    int y;
    ...
    C(y);
    ...
}
void B(float r) {
    int s, t;
    ...
    A(s);
    ...
}
void C(int q) {
    ...
}
void main() {
    float p;
    ...
    B(p);
    ...
}
```

MAIN, B'yi çağırdıktan sonra Stack

`main` `calls` **B**



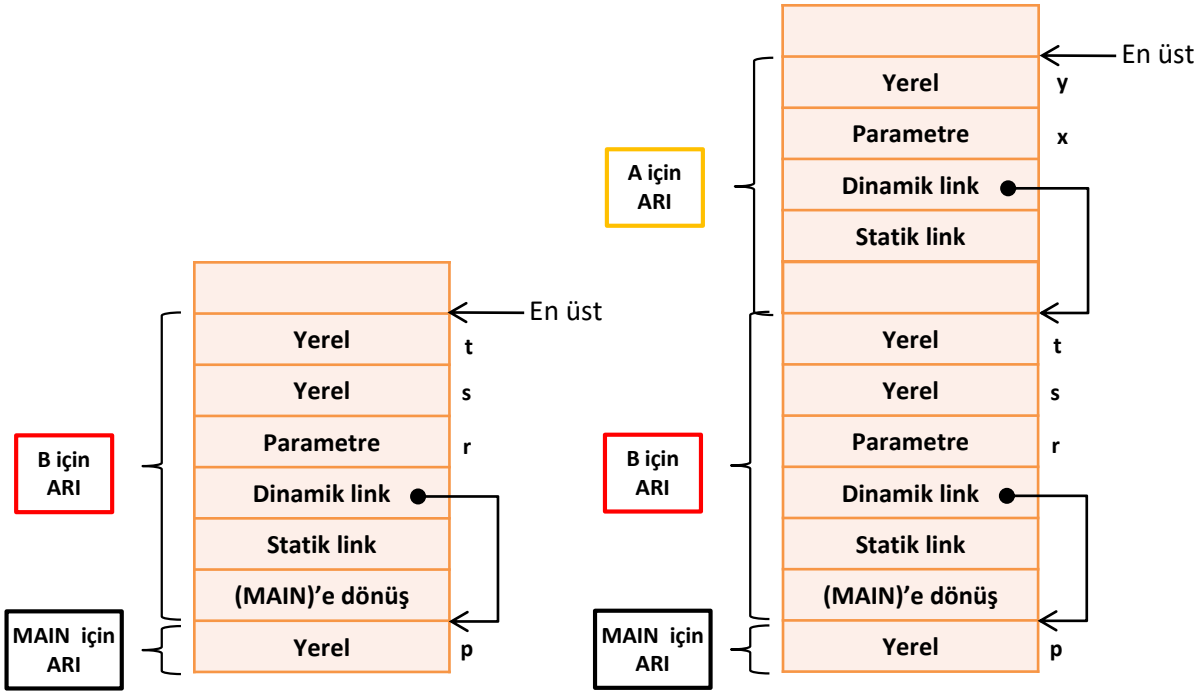

```
void main() {
  float p; ...
  B(p);...
}
```

```
void B(float r) {
  int s, t; ...
  A(s); ...
}
```

ARI = Etkinleştirme kayıt örneği (activation record instance)

B, A'yı çağırdıktan sonra Stack

main calls **B** calls **A**

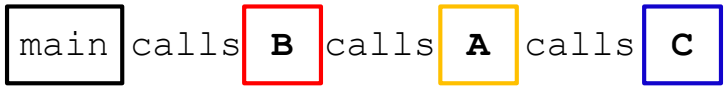



```
void main() {
  float p; ...
  B(p);...
}
```

```
void B(float r) {
  int s, t; ...
  A(s); ...
}
```

```
void A(int x) {
  int y; ...
  C(y); ...
}
```

A, C'yi çağırdıktan sonra Stack




```
void main() {
  float p; ...
  B(p);...
}
```

```
void B(float r) {
  int s, t; ...
  A(s); ...
}
```

```
void A(int x) {
  int y; ...
  C(y); ...
}
```

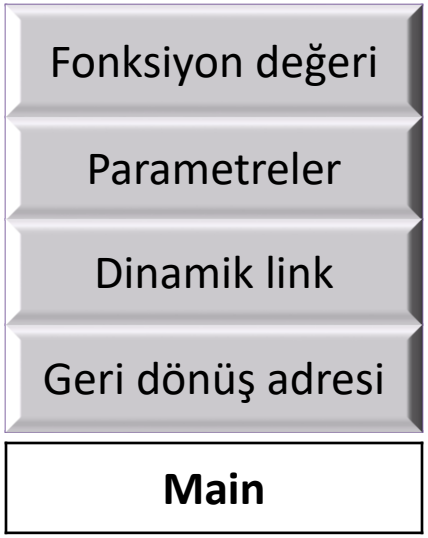
```
void C(int q) {
  ...
}
```

Altprogramların gerçekleştirimi

- Herhangi bir anda yığıttaki dinamik linklerin toplamına çağrı zinciri (call chain) denir.
- Yerel değişkenlere, etkinleştirme kaydının başlangıcına göre bağıl konumlarından (offset) erişilir. Buna yerel bağıl konum denir (local_offset).
- Yerel değişkenlerin yerel bağıl konumları derleyici tarafından belirlenir.

Özyinelemede Aktivasyon Kayıtları

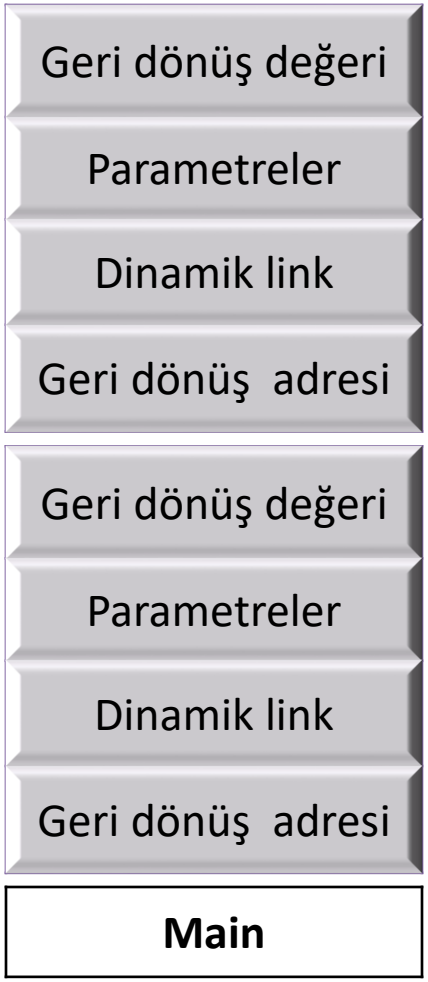
Bir önceki örnekte kullanılan etkinleştirme kaydı (activation record) özyinelemeli fonksiyonlarda da kullanılabilir



3

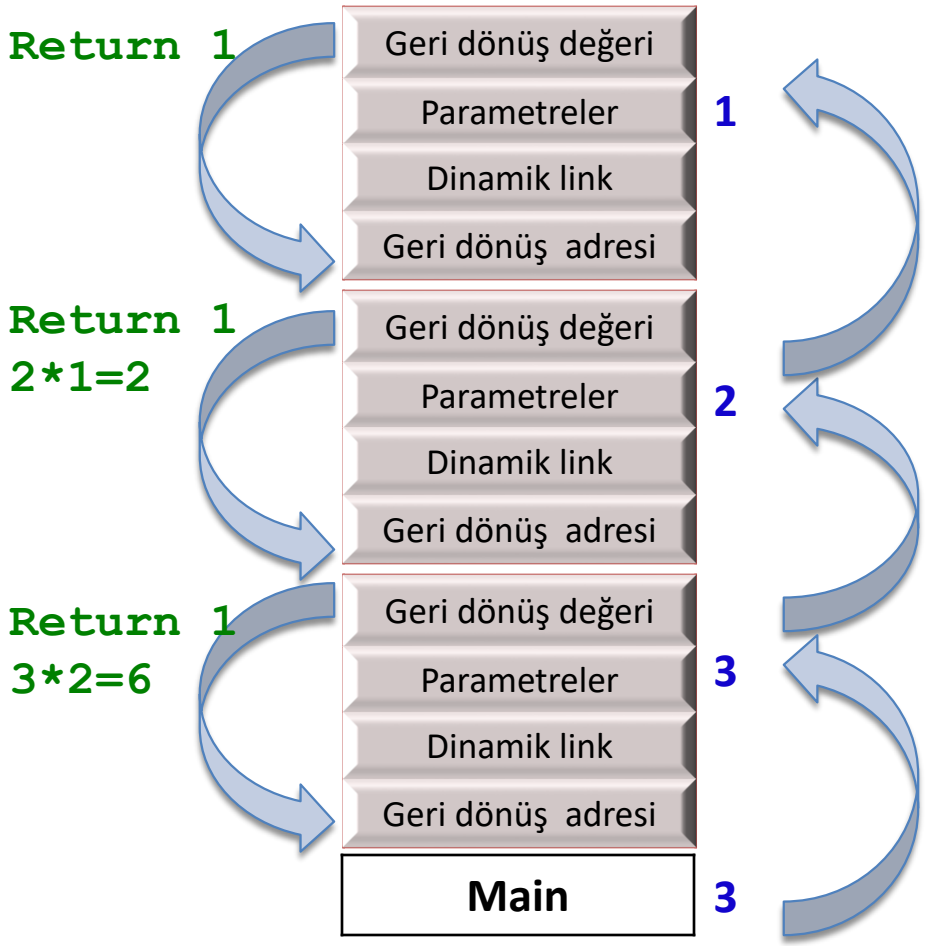
```
int factorial(int n) {
  if (n <= 1)
    return 1;
  else
    return (n * factorial(n-1));
}
void main() {
   int value;
   value = factorial(3);
}
```

Factorial için Aktivasyon Kayıtları




```
int factorial(int n) {
  if (n <= 1)
    return 1;
  else
    return (n * factorial(n-1));
}
void main() {
   int value;
   value = factorial(3);
}
```

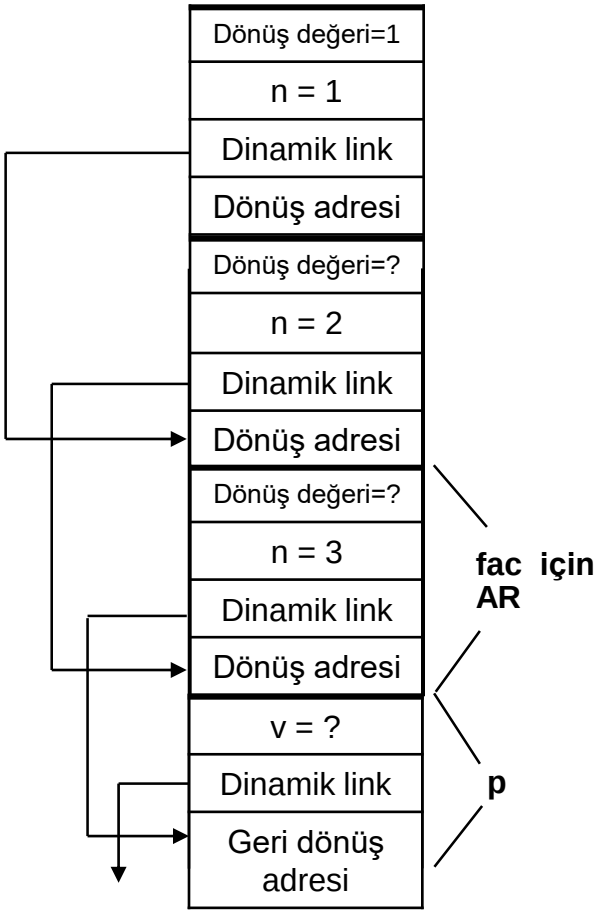
Factorial için Aktivasyon Kayıtları




```
int factorial(int n) {
  if (n <= 1)
    return 1;
  else
    return (n * factorial(n-1));
}
void main() {
    int value;
    value=factorial(3);
}
```

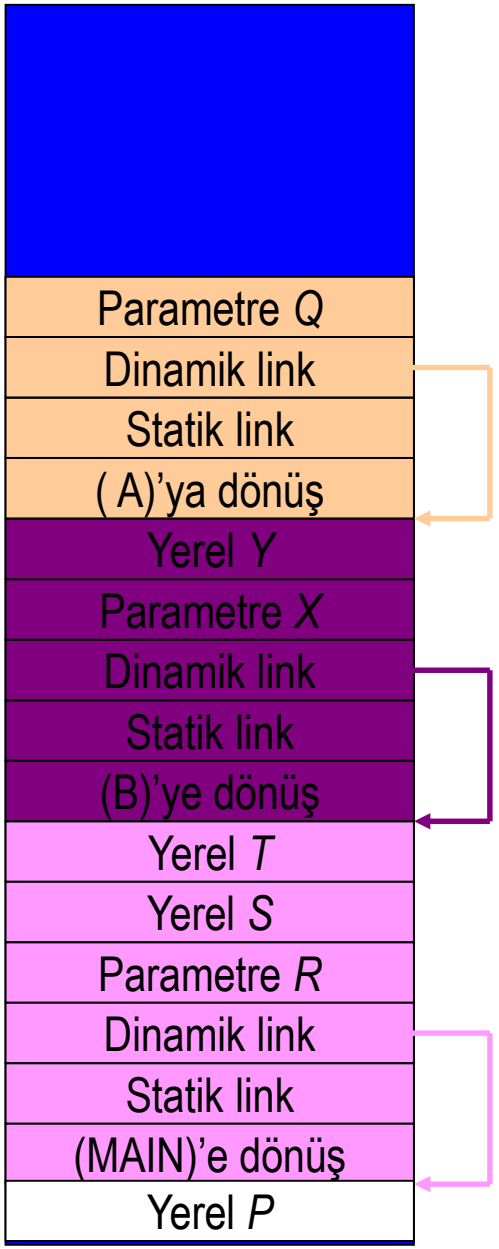
Faktoriyel Programı


```
program p;
  var v : int;
  function fac(n: int): int;
  begin
    if n <= 1 then
      fac := 1
    else
      fac := n * fac(n - 1);
  end;
begin
  v := fac(3);
  print(v);
end.
```



Yerel başvuru örneği

```
Program MAIN_1;
   var P : real;
   procedure A (X : integer);
      var Y : boolean;
      procedure C (Q : boolean);
         begin  {C}
         …
         end;  {C}
      begin {A}
      …
      C(Y);
      …
      end; {A}
   procedure B (R : real);
      var S, T :integer;
      begin {B}
      …
      A(S);
      …
      end; {B}
   begin {MAIN_1}
   …
   B(P);
   …
   end.  {MAIN_1}
```

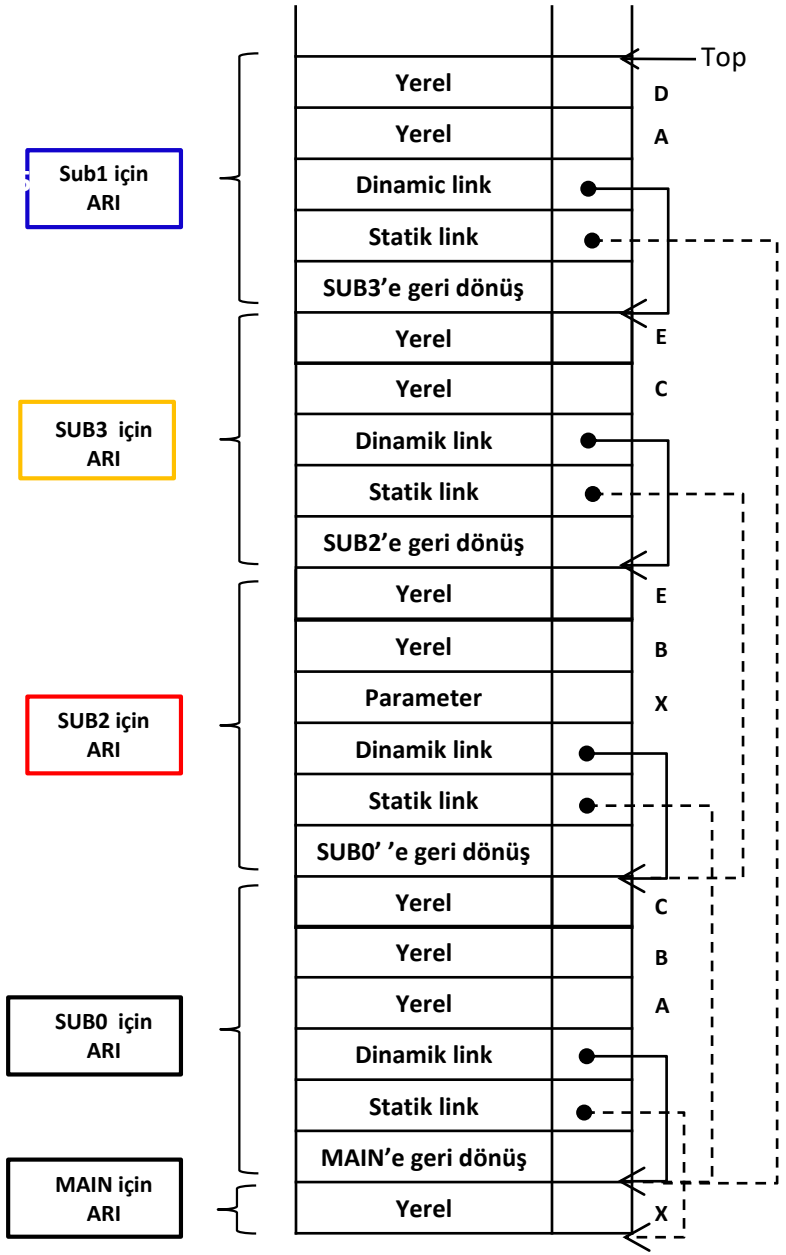


Özyineleme örneği

```
int factorial (int n){
 If (n <=1)
   return 1;
  else return (n* factorial (n-1));
}
void main(){
  int value;
  value = factorial (3);
}
```

İç içe altprogramlar(Nested subprograms)

- Bazı programlama dilleri (Fortran 95, Ada, JavaScript) yığıt dinamik yerel değişkenler (stack-dynamic local variables) kullanırlar ve iç içe altprogramlara izin verirler.
- Kural: Yerel olarak erişilmeyen tüm değişkenler yığıt bellekte aktif bir etkinleştirme kaydı içinde bulunurlar.
- Yerel olmayan değişken referansı bulma işlemi:
  1. Doğru etkinleştirme kaydını bul.
  2. Etkinleştirme kaydında değişkenin bağıl konumunu bul.




```
program MAIN;
  var X : integer;
  procedure SUB0;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
    begin { SUB1 }
      A := B + C;                          <------------1
    end; { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
      begin { SUB3 }
        SUB1;
        E := B + A;                        <--------2
      end; { SUB3 }
    begin { SUB2 }
      SUB3;
      A := D + E;                          <-----------3
    end; { SUB2 }
  begin { SUB0 }
    SUB2(7);
  end; { SUB0 }
begin
 SUB0;
end. { MAIN }
```

MAIN

MAIN çağırır SUB0

SUB0 çağırır SUB2

SUB2 çağırır SUB3

SUB3 çağırır SUB1