* Within an application server address space, many application programs run under a single server program.
* An error in one of these application programs can cause it to overwrite the code or data of the other application programs or of the server program itself.
* A **subspace** is identical image of the address space, but with access to limited parts of the address space.
  - It has the *same* ASID as the address space.
  - Unlike address spaces, data spaces, and hiperspaces, subspaces are *not* separate physical entities.
  - Subspaces do not occupy additional areas of storage; they are logical views of the programs and data within the address space.

  NOTE: The first exploiter of this was **CICS**. CICS uses subspaces to isolate transactions from each other to prevent one transaction from overwriting storage that belongs to another transaction. This increases the reliability and availability of CICS. This complements the public storage key (PSK) protection for CICS.
* **Subspaces are exploited by other environments** providing a means of limiting the application server address space storage that an application program can reference, thus limiting the damage an application program error can do within the application server address space.
  - A subspace is a specific range of storage in the private area of an address space, designed to limit the storage a program can reference.
  - A program that is associated with a subspace cannot reference some of the private area storage outside of the subspace storage range; the storage is "protected" from the program.
  - Whether a given range of private area storage is protected from a program associated with a subspace depends on whether the storage state is:
    1. Eligible to be assigned to a subspace (or **"subspace-eligible"**)
    2. Assigned to a subspace
    3. Not eligible to be assigned to a subspace.
  - You control these storage "states" through the IARSUBSP macro.

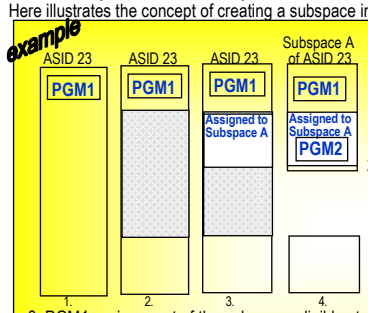
  NOTE: Storage outside of the address space's *private area* is not affected by subspaces (see #52 zTidBits (Storage).
* A program running in an address space can reference all of the storage associated with that address space which is known as **full address space addressability.**
  - A program that runs in an address space that owns subspaces also has full address space addressability, until it issues an instruction to limit the storage it can reference.
  - In an address space that owns subspaces, issuing the BRANCH IN SUBSPACE GROUP (**BSG**) instruction controls whether a program runs with full or limited address space addressability.
  - BSG is a hardware instruction that enables users to associate a program to a subspace

    (does not actually perform a branch into the subspace even though it contains BRANCH in the name).

  NOTE: A program running with limited addressability is said to be **running in a subspace.**
* The facility is designed to be used by a server address space that contains many applications running under the control of a single task (TCB), in order to prevent *any* given application program from modifying storage that belongs to one of the other application programs.
  - A unit of work can reference views of storage within the subspace.

  IMPORTANT A program running in a subspace can reference storage that is assigned to its *own* subspace and storage that is not eligible to be assigned to a subspace. In other words, a subspace allows a program running in it to reference *all* of the storage associated with the address space except the private area storage that is eligible to be assigned to a subspace or assigned to another subspace.
* When storage is not eligible to be assigned to a subspace and not already assigned to a subspace, it can be referenced by a program running in a subspace or a program running with full address space addressability.
  - This storage **can be referenced by all subspaces** as well as by programs running with full address space addressability.
  - An address space that owns subspaces is also called a **"base space"**.

  Here illustrates the concept of creating a subspace in address space (base) ASID 23 .



1. PGM1 is a program running with full address space addressability in address space ASID 23. ASID 23 owns no subspaces, and no storage eligible to be assigned to a subspace. PGM1 can reference all storage in the address space.
2. PGM1 makes the shaded area of storage eligible to be assigned to a subspace. The eligible storage has not been assigned to a subspace. PGM1 can reference the subspace-eligible storage because PGM1 is not running in a subspace.
3. PGM1 assigns part of the subspace-eligible storage to Subspace A. PGM1 can reference the subspace storage as well as the subspace-eligible storage because PGM1 is not running in a subspace.
4. PGM1 issues the BSG instruction, which passes control to PGM2 to run in Subspace A. PGM2 can reference the storage that is assigned to Subspace A, and storage in the address space that has not been made subspace-eligible. PGM2 cannot reference the subspace-eligible storage while PGM2 is running in the subspace.

* An address space can have many subspaces.
  - Each application program running simultaneously in an address space can run in its own subspace.
  - The subspace restricts a program running in it from referencing the storage assigned to other subspaces.
  - The number of subspaces is limited by the amount of unallocated private storage available in the address space and the amount of storage assigned to each subspace.
  - Ownership of a subspace is on a TCB basis (see #10 zTidBits Dispatchable UOW).

One factor that might influence your decision to use subspaces is the amount of virtual and central storage that the system requires to manage them. This storage overhead can affect system performance, but its benefits may out weight this.

It is most efficient to obtain storage early and create the number of subspaces needed for all application programs as part of application server initialization. Then, as a request for an application program's services is received, the server program assigns eligible storage to a subspace, runs the application program in the subspace, and disassociates the eligible storage from the subspace.

Before attempting to use subspaces, your program should ensure that the subspace is installed on your system. To test for the subspace, include the **CVT** in your program and check the CVTSUBSP bit. When bit is on, the subspace is available on your system. (**C**ommunication **V**ector **T**able)

‡ There are two types of **access lists entries** for addressability to address spaces. The two types differ from each other in the amount of authority-checking that the system does when a program in AR mode issues a data-referencing instruction and the data is in another address space.
An access list entry for an address space is either a **public entry** or a **private entry**, and a combination of both these types can be on the same DU-AL or PASN-AL.
A program can access the target address space through a public entry if it has (1) the access list entry that identifies the address space, and (2) the ALET for the entry.
A program can access the target address space through a private entry if it has (1) an access list entry that identifies the address space, (2) the ALET for the entry, and (3) the appropriate **extended authorization index (EAX)** value which zOS uses to control access to address spaces.

* A subspace is associated with *only* one address space and is *owned* by the task that creates it.
* An attached subtask or an SRB gets control with full address space addressability.
* A subspace has an access list entry ‡ (called an "**entry**") associated with it.
* After a program creates a subspace, it adds the entry to the dispatchable unit access list (DU-AL) associated with the task the program runs under.

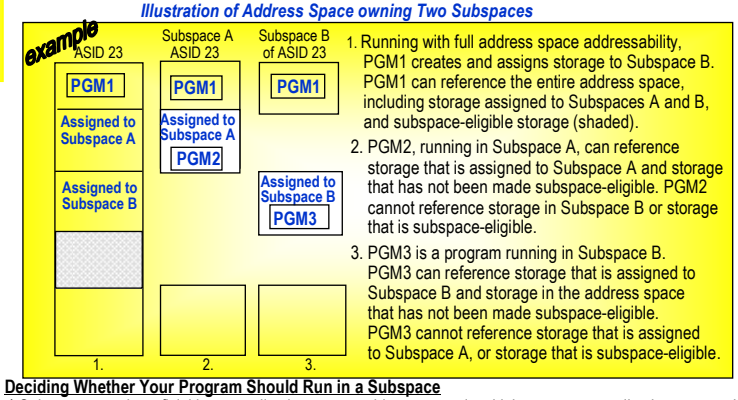
  NOTE: A dispatchable unit access list (DU-AL) — the access list that is associated with a work unit (a TCB or SRB), where each work unit has one DU-AL; See z/OS' MVS Programming Assembler Services Guide
* A program does not have to be in **Access Register (AR) mode** to use a subspace, although it can be.

  NOTE: An AR is a hardware register that a program uses to identify an address space or a data space. Each system has 16 ARs, numbered 0 through 15, and they are paired one-to-one with the 16 general purpose registers (GPRs).

Access Registers	0	1		Identify address spaces or data spaces			14	15
General Purpose Registers	0	1	Identify locations within an address space or data space				14	15

* A program can toggle between running in a subspace and running with full address space addressability by issuing the BSG instruction.
* Since an address space can have many subspaces; each application program running simultaneously in an address space can run in its own subspace.
* The subspace restricts a program running in it from referencing the storage assigned to other subspaces.
* Figure illustrates the concept of multiple subspaces by adding another subspace to address space ASID 23.

Illustration of Address Space owning Two Subspaces



1. Running with full address space addressability, PGM1 creates and assigns storage to Subspace B. PGM1 can reference the entire address space, including storage assigned to Subspaces A and B, and subspace-eligible storage (shaded).
2. PGM2, running in Subspace A, can reference storage that is assigned to Subspace A and storage that has not been made subspace-eligible. PGM2 cannot reference storage in Subspace B or storage that is subspace-eligible.
3. PGM3 is a program running in Subspace B. PGM3 can reference storage that is assigned to Subspace B and storage in the address space that has not been made subspace-eligible. PGM3 cannot reference storage that is assigned to Subspace A, or storage that is subspace-eligible.

Deciding Whether Your Program Should Run in a Subspace

* Subspaces are beneficial in an application server address space in which numerous applications run under a single task within the address space.
* Using subspaces requires few or no changes to the application programs.
* Using subspaces *does* require additional code in the server program.

Benefits of Subspaces

* The use of subspaces can protect the server and application programs in an address space.
* In addition, subspaces can help you to identify where in the address space an error has occurred.
* Using subspaces in an application server address space protects the server program and its data.
  - Subspaces reduce the number of failures in the server program by protecting it from the errors of other programs in the address space.
* Using subspaces in an application server address space also protects the applications, similar to the way programs are protected by running in separate address spaces.
  - By preventing applications from overwriting each other's code & data, subspaces increase the reliability of the applications
  - An IPCS diagnostic report and trace functions can help you to identify where in the address space an error has occurred.
  - An ABEND dump can help you to identify that an error resulted from a prohibited storage reference.
  - When requested by a program running in a subspace, an ABEND dump contains *only* the storage that the program is allowed to reference and therefore easier analysis and root cause.

Running a Program in a Subspace

* A program running in a subspace will abnormally end if it attempts to reference storage that is assigned to another subspace or storage that is eligible to be assigned to another subspace, but is not assigned.

  NOTE: Aside from these restrictions, a program running in a subspace can reference the *same storage* that a program running with full address space addressability can reference.

Subspaces have the following limitations:

* They *do not* provide protection against deliberate attempts to overwrite code.
* To ensure that subspace storage is protected, the system abnormally ends a program that:
  - Attempts to reference storage to which it does not have addressability
  - Provides incorrect information on the IARSUBSP macro.
* Coding additional recovery routines for your programs *might* be required.
* An unauthorized application program running in a subspace cannot add more storage to the subspace.
* If an application program requires *more* subspace storage, the server program *must obtain* subspace-eligible storage for the application.