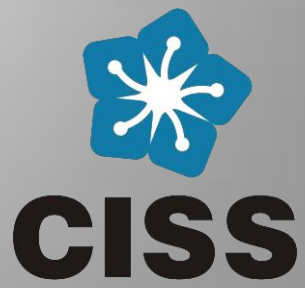
Kernel Pool Overflow:
от Windows XP до Windows 8

Никита Тараканов

CISS Research Team

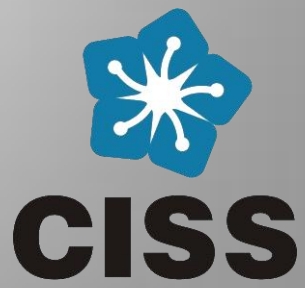
Kernel Pool

- Используется для работы с динамической памятью в пространстве ядра
- Память доступна всем компонентам, работающим в пространстве ядра
- Различные типы памяти
- Каждый Pool описан структурой nt!_POOL_DESCRIPTOR



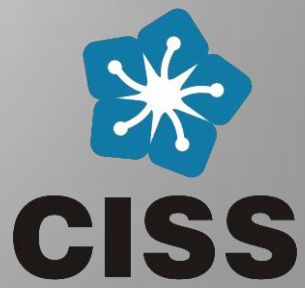
Типы памяти

- Определены в nt!_POOL_TYPE
- Много разновидностей – с каждым новым выпуском Windows кол-во увеличивается
- Три основных типа: Non-Paged, Paged, Session



nt!_POOL_TYPE(7 SP1)

```
typedef enum _POOL_TYPE{
    NonPagedPool                    = 0 /*0x0*/,
    PagedPool                       = 1 /*0x1*/,
    NonPagedPoolMustSucceed         = 2 /*0x2*/,
    DontUseThisType                 = 3 /*0x3*/,
    NonPagedPoolCacheAligned        = 4 /*0x4*/,
    PagedPoolCacheAligned           = 5 /*0x5*/,
    NonPagedPoolCacheAlignedMustS   = 6 /*0x6*/,
    MaxPoolType                     = 7 /*0x7*/,
    NonPagedPoolSession             = 32 /*0x20*/,
    PagedPoolSession                = 33 /*0x21*/,
    NonPagedPoolMustSucceedSession  = 34 /*0x22*/,
    DontUseThisTypeSession          = 35 /*0x23*/,
    NonPagedPoolCacheAlignedSession = 36 /*0x24*/,
    PagedPoolCacheAlignedSession    = 37 /*0x25*/,
    NonPagedPoolCacheAlignedMustSSession = 38 /*0x26*/
```



nt!_POOL_TYPE(8 Dev. Prev.)

- Новые типы Pool'a – **Not Executable**:

1. NonPagedPoolNx = 512 /*0x200*/
2. NonPagedPoolNxCacheAligned = 516 /*0x204*/
3. NonPagedPoolSessionNx = 544 /*0x220*/

CISS

Non-Paged Pool

- **Не**выгружаемая системная память
- Доступна на любом уровне IRQ
- Количество Non-Paged Pool определено в nt!ExpNumberOfNonPagedPool и зависит от количества нодов nt!KeNumberNodes
- Создаётся в nt!InitializePool
- Указатель хранится в nt!PoolVector[0]
- Указатель хранится в массиве nt!ExpNonPagedPoolDescriptor(NUMA)

CISS

Paged Pool

- Выгружаемая системная память
- Доступна на PASSIVE,APC уровнях IRQ
- Количество Paged Pool определено в nt!ExpNumberOfPagedPool
- Создаётся в nt!InitializePagedPool
- Paged Pool дескрипторы хранятся в nt!ExpPagedPoolDescriptor с 1-го индекса
- Один допольнительный Paged Pool дескриптор хранится в nt!ExpPagedPoolDescriptor [0]

CISS

Session Paged Pool

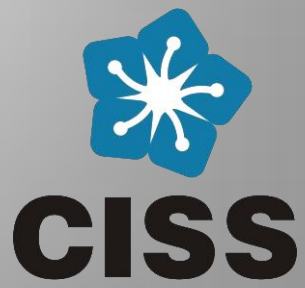
- Выгружаемая системная память, используется в пространстве сессии
- Уникальна для каждого пользователя
- Создаётся в nt!MiInitializeSession

Kernel Pool Descriptor

- typedef struct _POOL_DESCRIPTOR {
- /*0x000*/ **enum _POOL_TYPE PoolType**;
- /*0x004*/ **ULONG32 PoolIndex**;
- /*0x008*/ ULONG32 RunningAllocs;
- /*0x00C*/ ULONG32 RunningDeAllocs;
- /*0x010*/ ULONG32 TotalPages;
- /*0x014*/ ULONG32 TotalBigPages;
- /*0x018*/ ULONG32 Threshold;
- /*0x01C*/ VOID* LockAddress;
- /*0x020*/ **VOID* PendingFrees**;
- /*0x024*/ LONG32 PendingFreeDepth;
- /*0x028*/ **struct _LIST_ENTRY ListHeads[512]**;
- }POOL_DESCRIPTOR, *PPOOL_DESCRIPTOR;

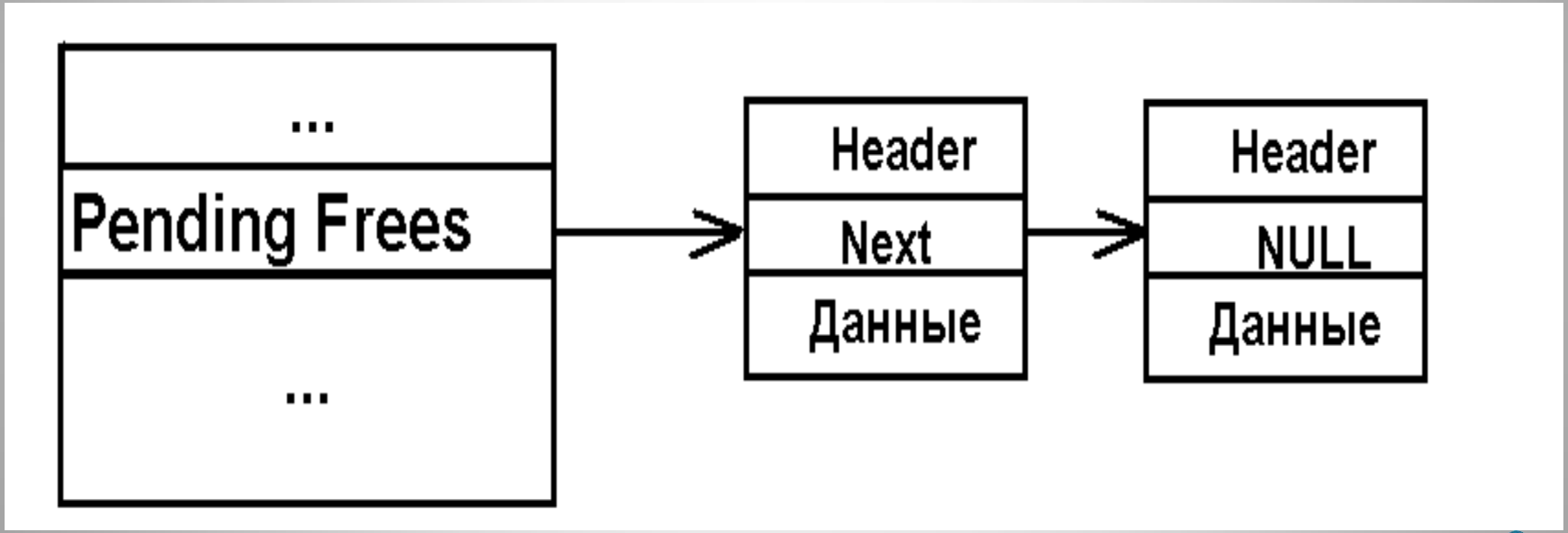
Kernel Pool Descriptor

- PoolType – тип памяти(Non-Paged, Paged…)
- PoolIndex – индекс Pool Descriptor’a в массиве дескрипторов
- PendingFrees – односвязный список чанков, ожидающих освобождения
- ListHeads – 512 двусвязных списков свободных чанков памяти



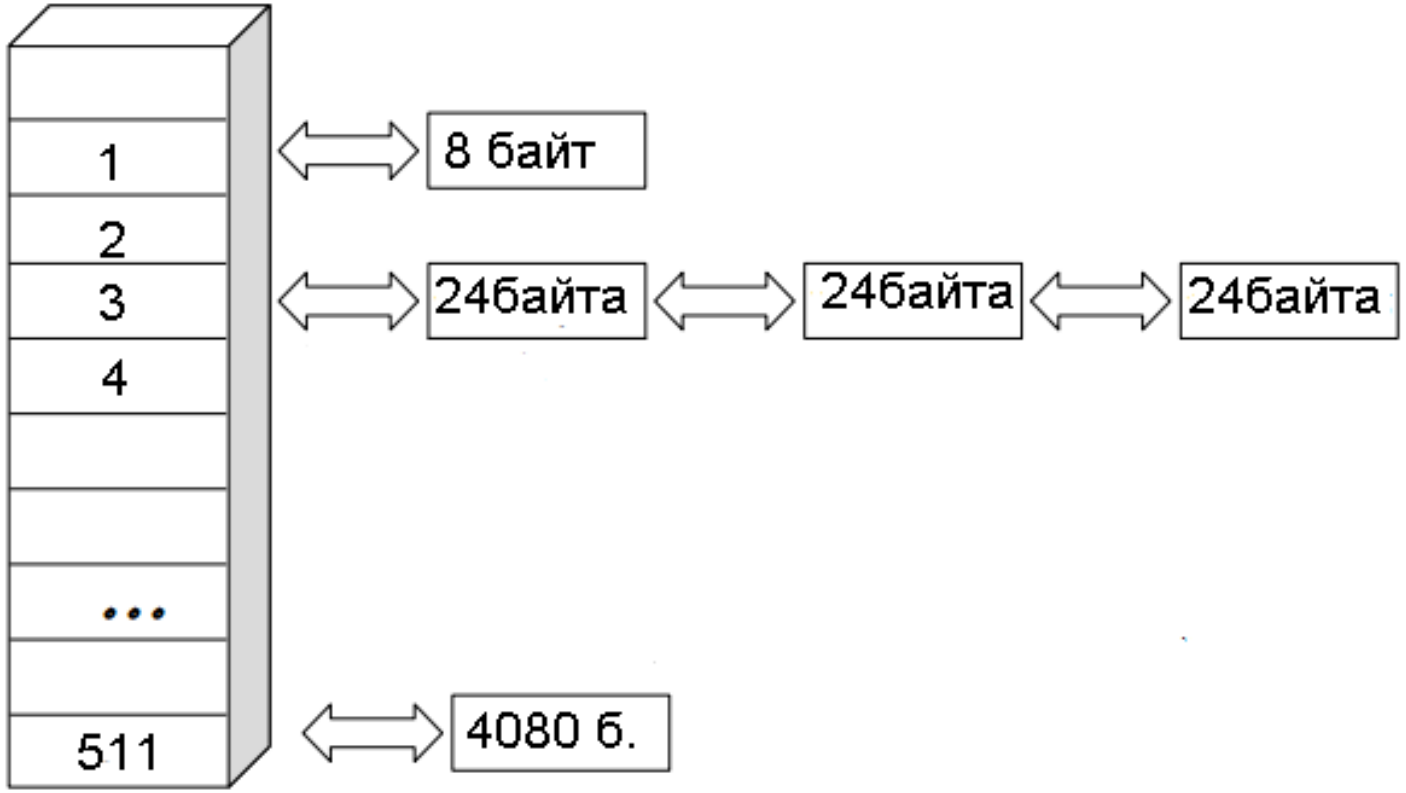
PendingFrees

- Pool Descriptor PendingFrees



CISS

ListHeads

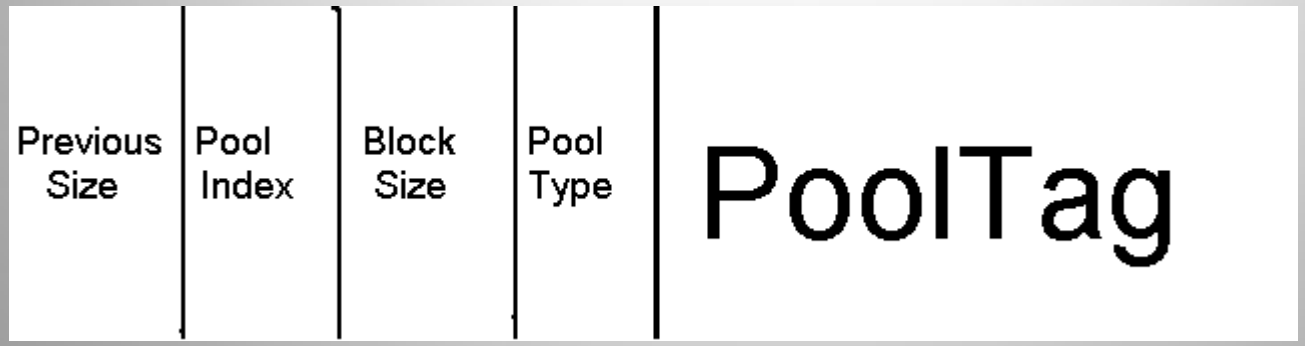


CISS

Описатель чанка памяти(x86)

typedef struct _POOL_HEADER {

- /*0x000*/ UINT16 PreviousSize : 9;
- /*0x000*/ UINT16 PoolIndex : 7;
- /*0x002*/ UINT16 BlockSize : 9;
- /*0x002*/ UINT16 PoolType : 7;
- /*0x004*/ ULONG32 PoolTag;
- }POOL_HEADER, *PPOOL_HEADER;

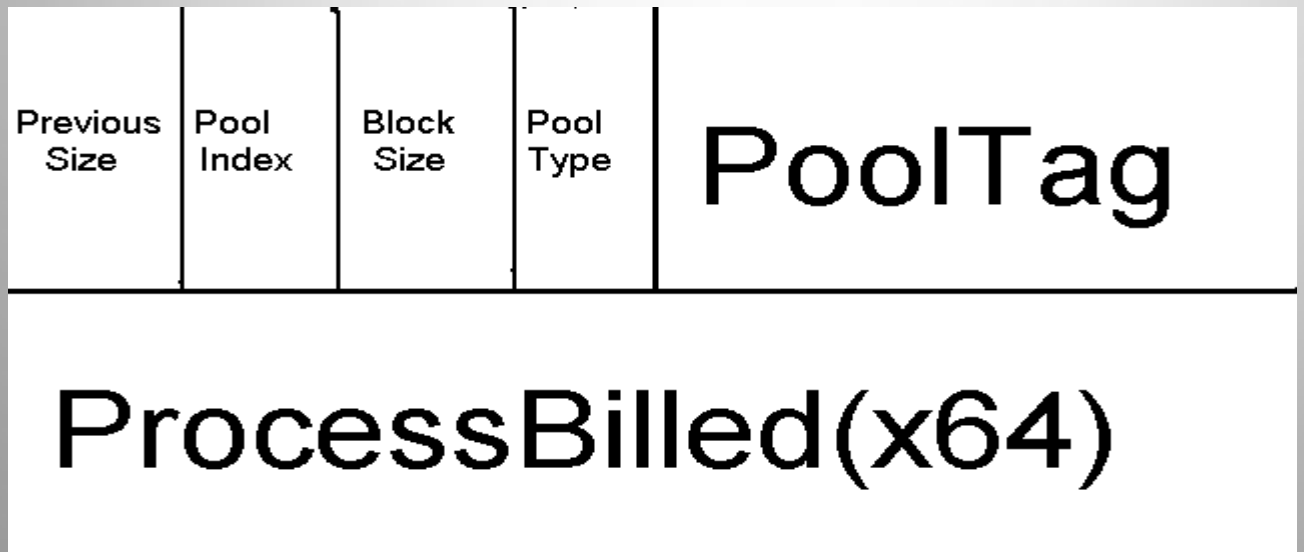


CISS

Описатель чанка памяти(x64)

typedef struct _POOL_HEADER {

- /*0x000*/ UINT16 PreviousSize : 9;
- /*0x000*/ UINT16 PoolIndex : 7;
- /*0x002*/ UINT16 BlockSize : 9;
- /*0x002*/ UINT16 PoolType : 7;
- /*0x004*/ ULONG32 PoolTag;
- /*0x008*/ EPROCESS *ProcessBilled;
-



CISS

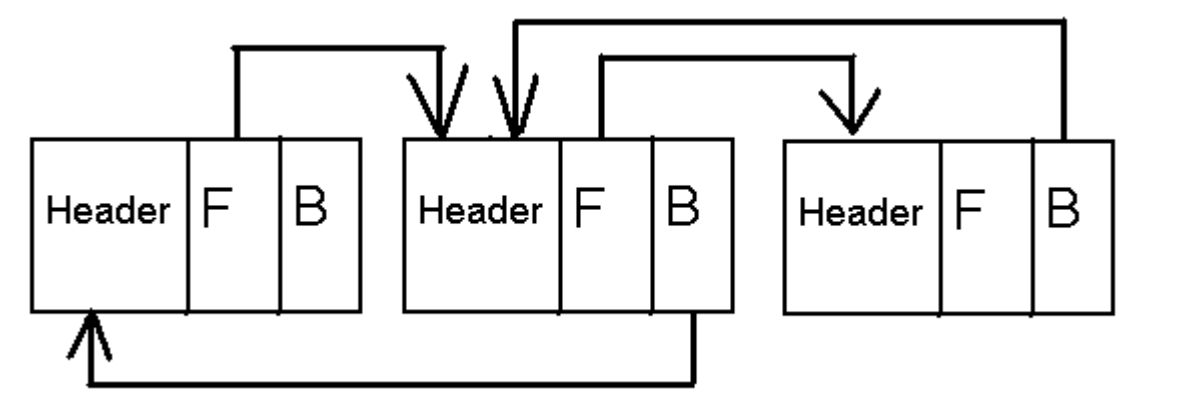
Описатель чанка памяти

- BlockSize – (кол-во байт + 0xf) >> 3
- PreviousSize – BlockSize предыдущего чанка
- PoolIndex – индекс соответствующего Pool’a
- PoolType – 0 если свободный
- PoolTag – 4 символа обозначающих какой код осуществил аллокацию
- ProcessBilled(x64) – указатель на EPROCESS(используется для Quota)

CISS

Освобождённый чанк памяти

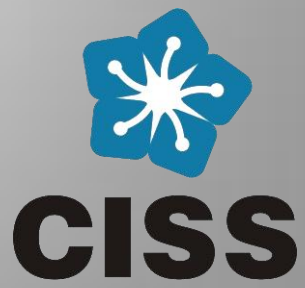
- Чанки находящиеся в ListHeads содержат 2 указателя(структура LINK_ENTRY) сразу после описателя



CISS

Lookaside Lists

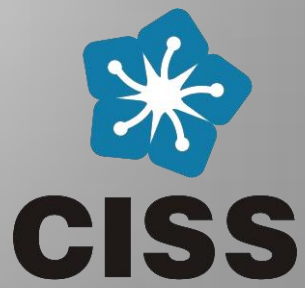
- Используются для быстрой (де)аллокации малых объёмов памяти(до 256 байт)
- Раздельные списки для Non-Paged, Paged памяти
- Определён структурой **nt!GENERAL_LOOKASIDE_POOL**



nt!GENERAL_LOOKASIDE_POOL

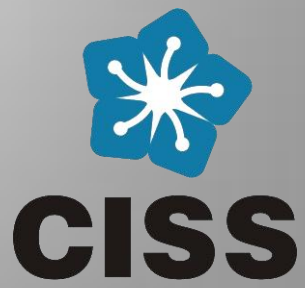
typedef struct _GENERAL_LOOKASIDE_POOL{

- union{
- /*0x000*/ union _SLIST_HEADER ListHead;
- /*0x000*/ struct _SINGLE_LIST_ENTRY SingleListHead;};
- /*0x008*/ UINT16 Depth;
- /*0x00A*/ UINT16 MaximumDepth;
- /*0x00C*/ ULONG32 TotalAllocates;
- union{
- /*0x010*/ ULONG32 AllocateMisses;
- /*0x010*/ ULONG32 AllocateHits;};
- /*0x014*/ ULONG32 TotalFrees;
- union{
- /*0x018*/ ULONG32 FreeMisses;
- /*0x018*/ ULONG32 FreeHits;};



nt!GENERAL_LOOKASIDE_POOL

```
/*0x01C*/    enum _POOL_TYPE Type;
/*0x020*/    ULONG32     Tag;
/*0x024*/    ULONG32     Size;
/*0x028*/    VOID* Allocate;
/*0x02C*/    VOID* FreeEx;
/*0x030*/    struct _LIST_ENTRY ListEntry;
/*0x038*/    ULONG32     LastTotalAllocates;
           union{
/*0x03C*/        ULONG32     LastAllocateMisses;
/*0x03C*/        ULONG32     LastAllocateHits;};
/*0x040*/    ULONG32     Future[2];
         }GENERAL_LOOKASIDE_POOL, *PGENERAL_LOOKASIDE_POOL;
```



Алгоритмы аллокации памяти

- Зависит от объёма и типа памяти
- Различные ф-ии: ExAllocatePool(WithTag, WithQuota и т.д.)
- Используются односвязные списки для малых(<256 байт) объёмов памяти
- Используются двусвязные списки для средних(256 < N < 4080) объёмов памяти
- Используется аллокатор страниц

CISS

ExAllocatePoolWithTag

- PVOID **ExAllocatePoolWithTag**(POOL_TYPE PoolType, SIZE_T NumberOfBytes, ULONG Tag)
- Если NumberOfBytes > 4080 – вызов nt!MiAllocatePoolPages/nt!ExpAllocateBigPool
- Если PoolType == Paged:
- Если BlockSize <= 32 – запрос Paged Lookaside Lists
- Если PoolType относится к Session и BlockSize <= 25 – запрос к Session Lookaside Lists

CISS

ExAllocatePoolWithTag

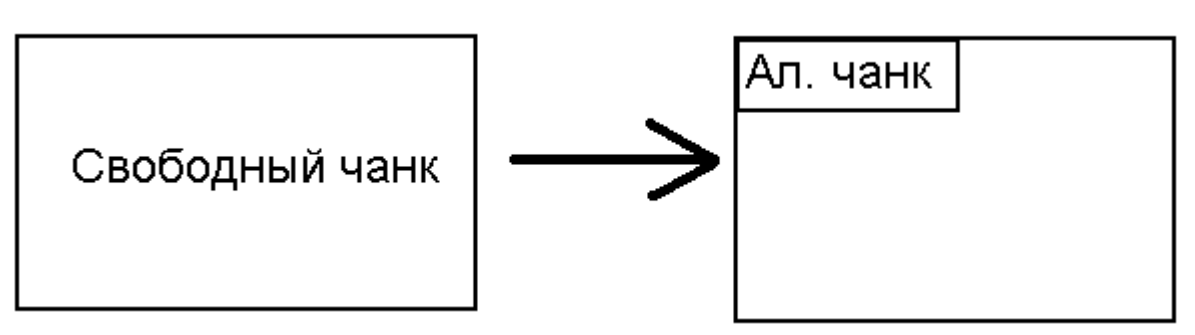
- Если PoolType == Non-Paged и BlockSize <= 32 – запрос Non-Paged Lookaside Lists
- Поиск первого не пустого ListHeads[n], BlockSize <= n < 512
- Если выбранный чанк больше – разделение
- Если чанк запрашиваемой длины не найден – расширение Pool’a(Вызов MiAllocatePoolPages)

Разделение чанка памяти

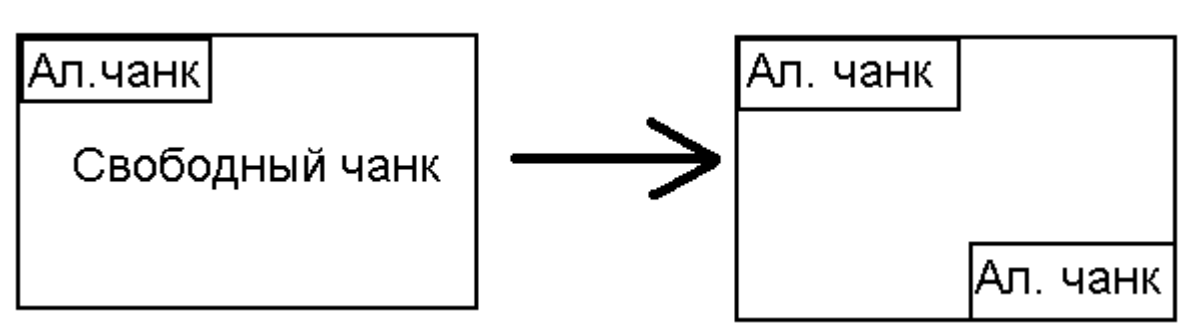
- Если алгоритм вернул чанк большего размера – происходит разделение
- Если чанк находится в начале страницы – выделить необходимую длину с начала чанка
- Если чанк находится не в начале страницы – выделить необходимую длину с конца чанка
- Оставшаяся часть возвращается в ListHeads

Разделение чанка памяти

- Чанк в начале страницы



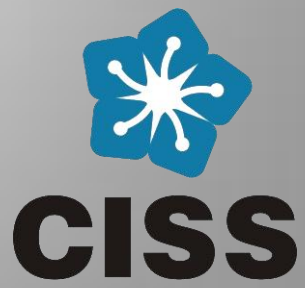
- Чанк **не** в начале страницы



CISS

Алгоритм деаллокации

- Различные ф-ии: **ExFreePool**(WithTag, WithQuota…)
- Алгоритм деаллокации “обрабатывает” чанк(описатель чанка) памяти – возвращает освобождённый чанк в соответствующий список: Lookaside(<256 байт), ListHeads<(4080 байт)
- Соединяет смежные чанки
- Освобождает страницы памяти



ExFreePoolWithTag

- VOID **ExFreePoolWithTag**(PVOID Address, ULONG Tag)
- Если Address выравнен по странице памяти – вызов nt!MiFreePoolPages
- Если BlockSize <= 32 – возврат в соответствующий (Non-Paged, Paged) Lookaside List
- Если PoolType относится к Session и BlockSize <= 25 – возврат в Session Lookaside List

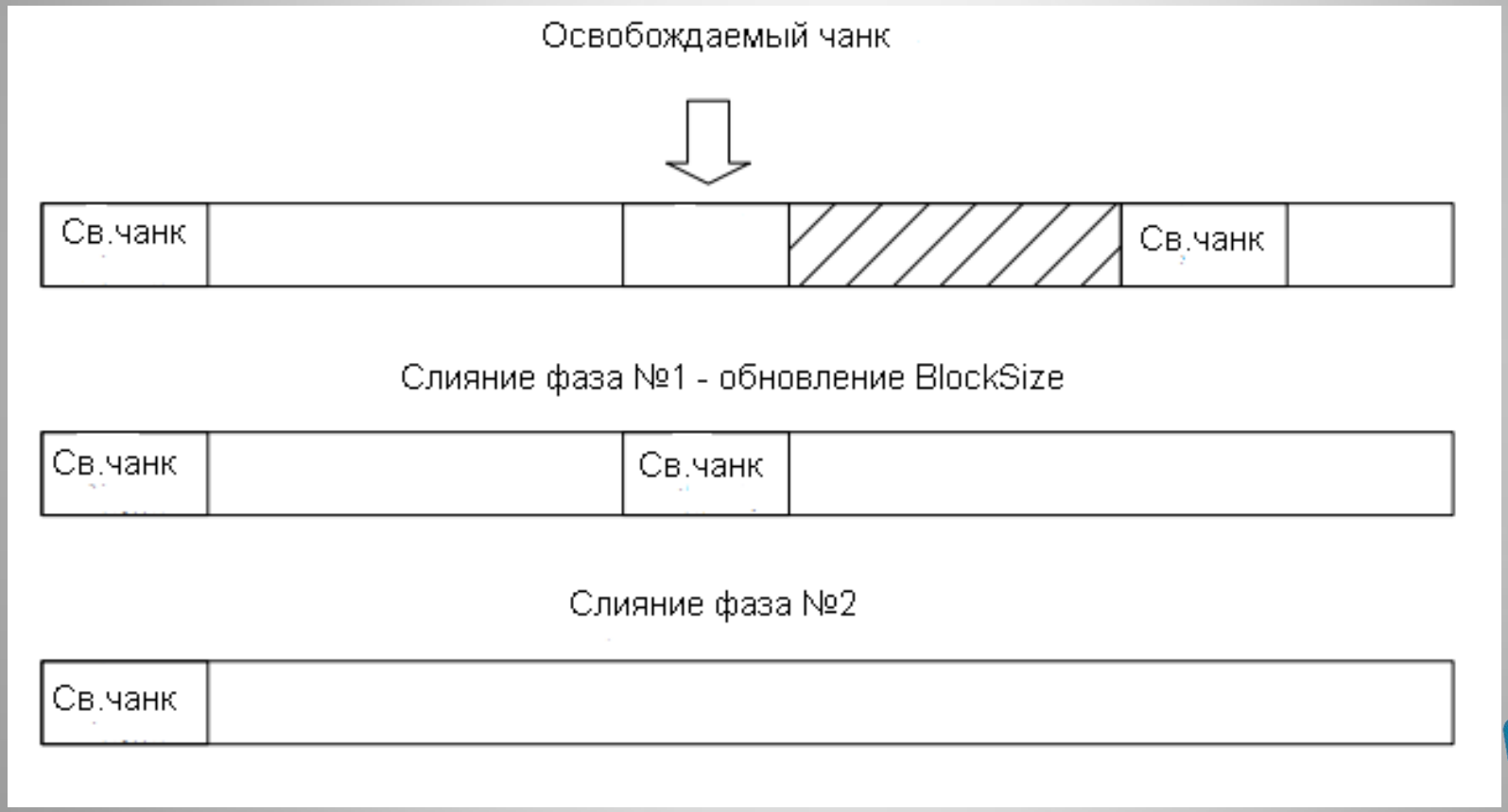
CISS

ExFreePoolWithTag

- Если установлен флаг DELAY_FREE (ExpPoolFlags&0x200) и кол-во чанков “ждущих” деаллокации >= 32 – вызов **nt!ExDeferredFreePool**
- Если следующий чанк свободный и не выравнен по странице памяти – соединить с текущим
- Если предыдущий чанк свободный – соединить с текущим
- Если результирующий чанк размером в страницу – вызвать **nt!MiFreePoolPages**
- Добавить чанк в соответствующий ListHeads

CISS

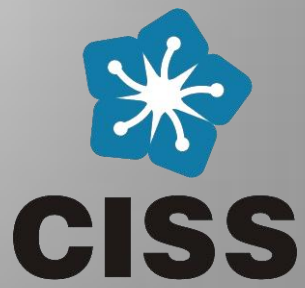
Слияние чанков



CISS

Generic атаки

- Generic атаки используют алгоритмы (де)аллокации памяти – манипулирование метаданными(описатель чанка и т.д.) – для получения стандартного условия write4 **arbitrary memory write**



Перезапись указателей (LIST_ENTRY)

- ListHeads содержат двусвязные списки свободных чанков, связанных между собой структурой LIST_ENTRY

```
RemoveEntryList(Entry) {
    PLIST_ENTRY Blink;
    PLIST_ENTRY Flink;
    Flink = Entry->Flink; //значение
    Blink = Entry->Blink;//адрес
    Blink->Flink = Flink; // *(адрес) = значение
    Flink->Blink = Blink; // *(значение+4) = адрес
```

Методы

1. Слияние со следующим чанком
2. Слияние с предыдущим чанком
3. Аллокация из ListHeads[n]

- Не работает с Windows 7 - из-за внедрения safe unlinking’a

Safe Unlinking

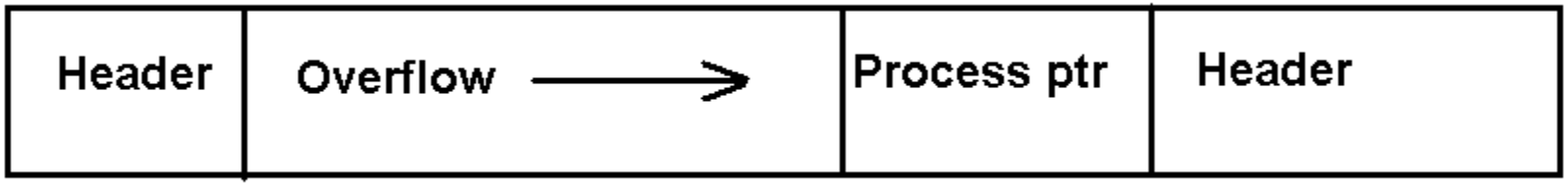
```
RemoveEntryList(Entry) {
    PLIST_ENTRY Blink;
    PLIST_ENTRY Flink;
    Flink = Entry->Flink;
    Blink = Entry->Blink;
    if(Blink->Flink != Flink->Blink) KeBugCheckEx();
    Blink->Flink = Flink;
    Flink->Blink = Blink;
```

Перезапись указателя Quota Process

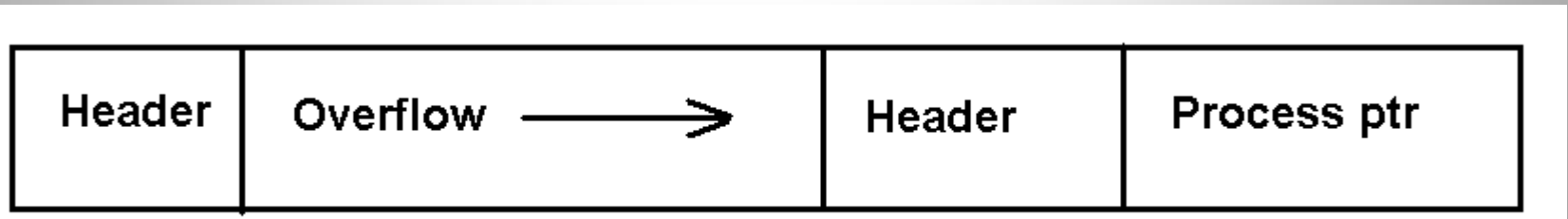
- Pool аллокации использующие Quota(ExAllocatePoolWithQuotaTag) хранят указатель на объект процесса
- При деалокации Quota освобождается и объект процесса разыменовывается
- Перезапись указателя ведёт к memory corruption(PspReturnQuota), arbitrary memory decrement(ObfDereferenceObject)
- Не работает с Windows 8 Developer Preview из-за внедрения nt! ExpPoolQuotaCookie

Перезапись указателя Quota Process

- x86



- x64



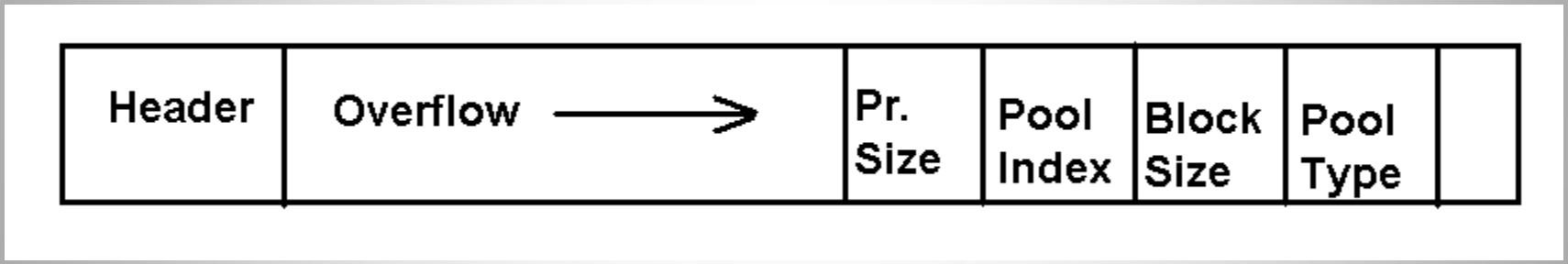
CISS

Перезапись PoolIndex

- PoolIndex описателя чанка обозначает индекс Pool'а в соответствующем массиве
- Для Paged Pool'а PoolIndex обозначает индекс в массиве nt!ExpPagedPoolDescriptor
- Для Non-Paged Pool'а PoolIndex обозначает индекс в массиве nt!ExpNonPagedPoolDescriptor, если nt!KeNumberNodes > 1

CISS

Перезапись PoolIndex



CISS

Перезапись PoolIndex

- Формирование невалидного PoolIndex ведёт к освобождению чанка в Pool Descriptor расположенному по нулевому адресу
- При возврате в контролируемый Pool Descriptor – происходит запись адреса освобождённого чанка по контролируемому адресу
- Не работает с Windows 8 Developer Preview – запрет на аллокацию по нулевому адресу

Generic атаки

- Требуют определённых условий (позиция чанков, тип памяти и т.д.)
- С каждой новой версией ОС – гайки закручиваются сильнее
- Для уязвимостей некоторого типа трудно применимы
- Для таких случаев требуется другой подход…

Custom атаки

- Перезапись не метаданных, а **данных** чанка

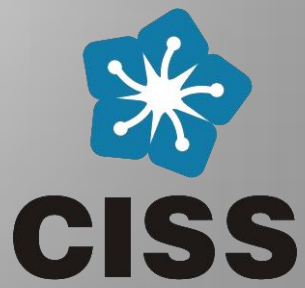
- Profit?

Custom атаки

- Наилучшие типы данных для перезаписи:

1. Указатели на функции
2. Структуры

- Где взять такие данные?!
- Ядерные объекты!



Custom атаки

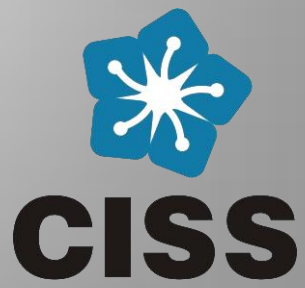
- Большинство ядерных объектов содержат указатели
- Интерфейсы для создания, изменения, чтения, удаления: Nt**Create**WorkerFactory, Nt**SetInformation**WorkerFactory, Nt**QueryInformation**WorkerFactory, Nt**Close**

Custom атаки

- Использование Kernel Pool Overflow для:
- Arbitrary Code Execution
- Arbitrary Memory Overwrite
- Arbitrary Memory Read(!)

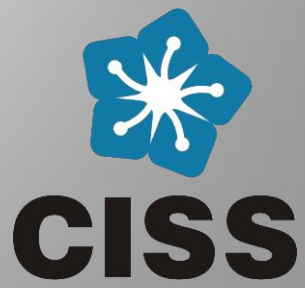
Custom атаки : Code Execution

- typedef struct _KTIMER{
- /*0x000*/ struct _DISPATCHER_HEADER Header;
- /*0x010*/ union _ULARGE_INTEGER DueTime;
- /*0x018*/ struct _LIST_ENTRY TimerListEntry;
- /*0x020*/ **struct _KDPC* Dpc;**
- /*0x024*/ ULONG32 Period;
- }KTIMER, *PKTIMER;



Custom атаки : Code Execution

```
typedef struct _KDPC {
/*0x000*/    UINT8          Type;
/*0x001*/    UINT8          Importance;
/*0x002*/    UINT16         Number;
/*0x004*/    struct _LIST_ENTRY DpcListEntry;
/*0x00C*/    VOID*          DeferredRoutine;
/*0x010*/    VOID*          DeferredContext;
/*0x014*/    VOID*          SystemArgument1;
/*0x018*/    VOID*          SystemArgument2;
/*0x01C*/    VOID*          DpcData;
        }KDPC, *PKDPC;
```



Custom атаки : Arbitrary Write

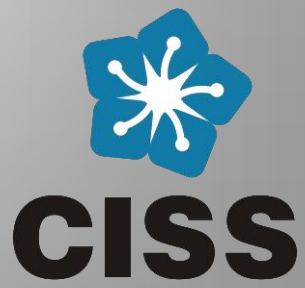
- NtSetInformationWorkerFactory

```
.text:004D08CD                  mov     ecx, [ebp+WorkerFactoryObject]
.text:004D08D0                  mov     eax, [ecx+4]    ; eax под контролем!
.text:004D08D3                  mov     eax, [eax+4]
.text:004D08D6                  test    edi, edi
.text:004D08D8                  jnz     short loc_4D08E3 ; Arbitrary memory write!
.text:004D08DA                  mov     edi, ds:_KeNumberProcessors
.text:004D08E0                  mov     ecx, [ebp+WorkerFactoryObject]
.text:004D08E3
.text:004D08E3 loc_4D08E3:                              ; CODE XREF: NtSetInformationWor
.text:004D08E3                  mov     [eax+1Ch], edi  ; Arbitrary memory write!
```

Custom атаки: Arbitrary read

- NtQueryInformationWorkerFactory

```
mov     eax, [esi+14h]  ; esi - Object
mov     eax, [eax+0B4h] ; eax - под контролем, arbitrary memory read
mov     [ebp+output], eax
```



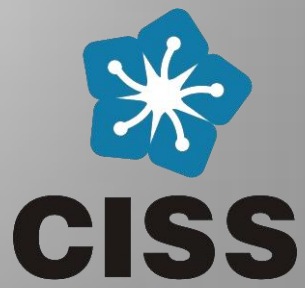
Kernel Pool Spray

- Временный VS постоянный(контролируемый)
- Временный – пример NtDeviceIoControlFile, ioctl METHOD_BUFFERED
- Постоянный – контроль над временем жизни, содержимым

CISS

Kernel Pool Spray

- Non-Paged: **ObCreateObject –** NtCreateTimer, NtCreateEvent, NtCreateWaitCompletionPacket и т.д.
- Paged: Unicode строки – свойства ядерных объектов

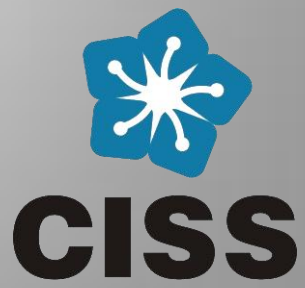


Kernel Pool Spray

- В Windows 7 появился специальный системный вызов для удобного манипулирования Kernel Pool’ом
- **NtAllocateReserveObject**

Kernel Pool Explotation Mitigations

- Максимальное кол-во проверок метаданных (аналог многочисленных защит в Userland Heap)
- Cookie для ядерных объектов?!



What's the heck!

- Dude, where is the demo???
- Where is the super private 0day?

Not this time folks!

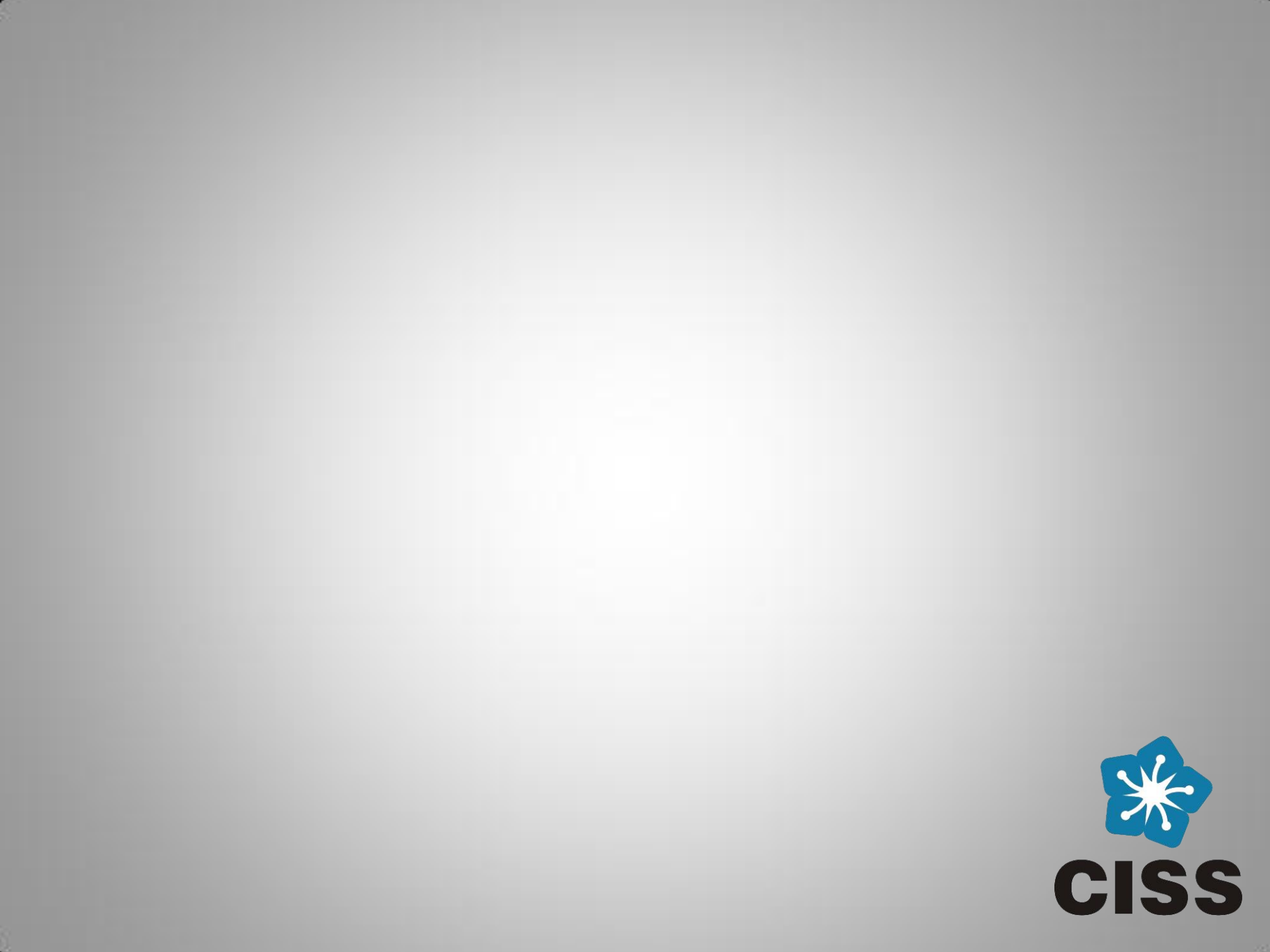
- Drop me an e-mail
- Or
- Wait **0-day time show this evening**

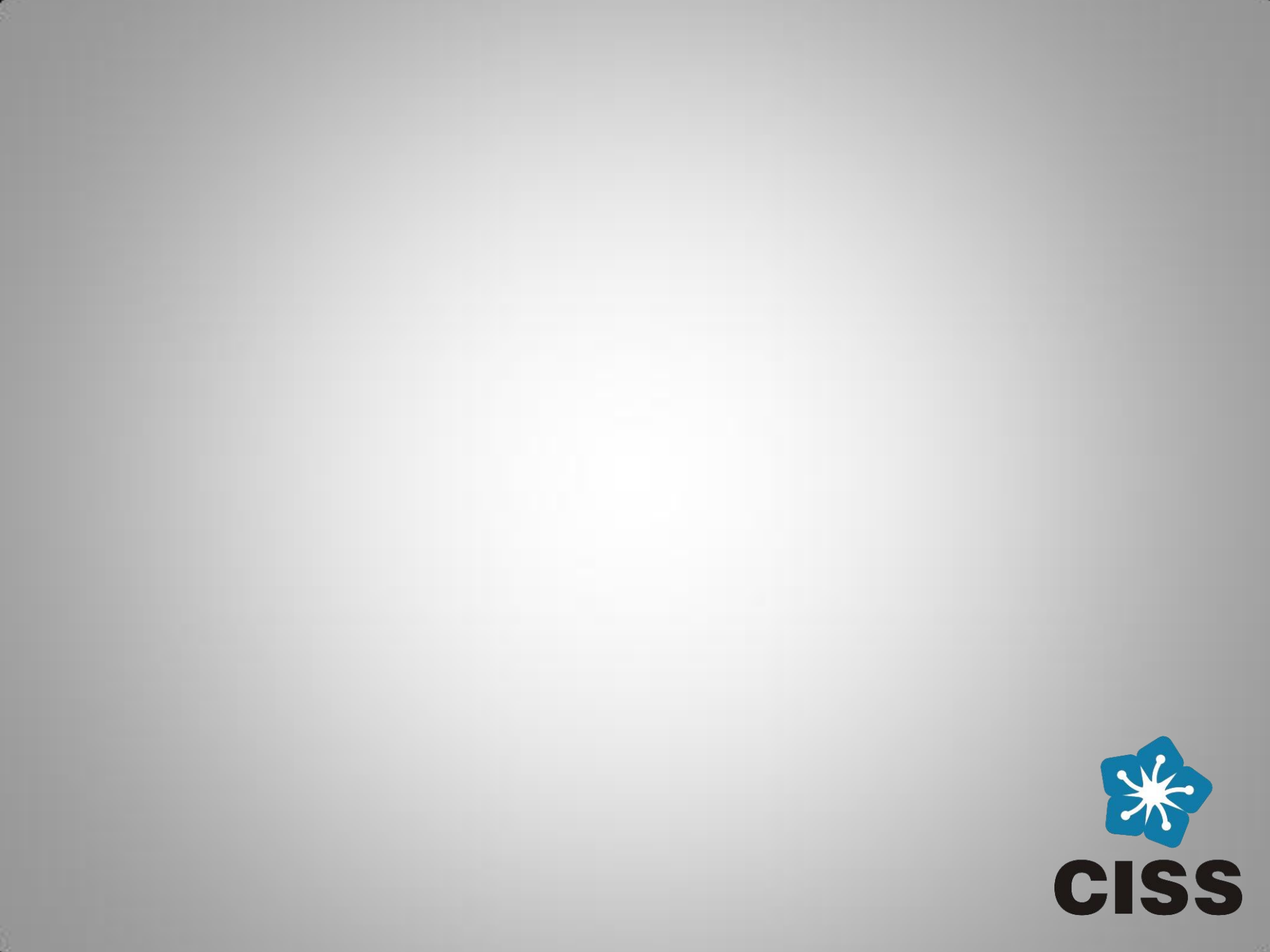
Вопросы!?

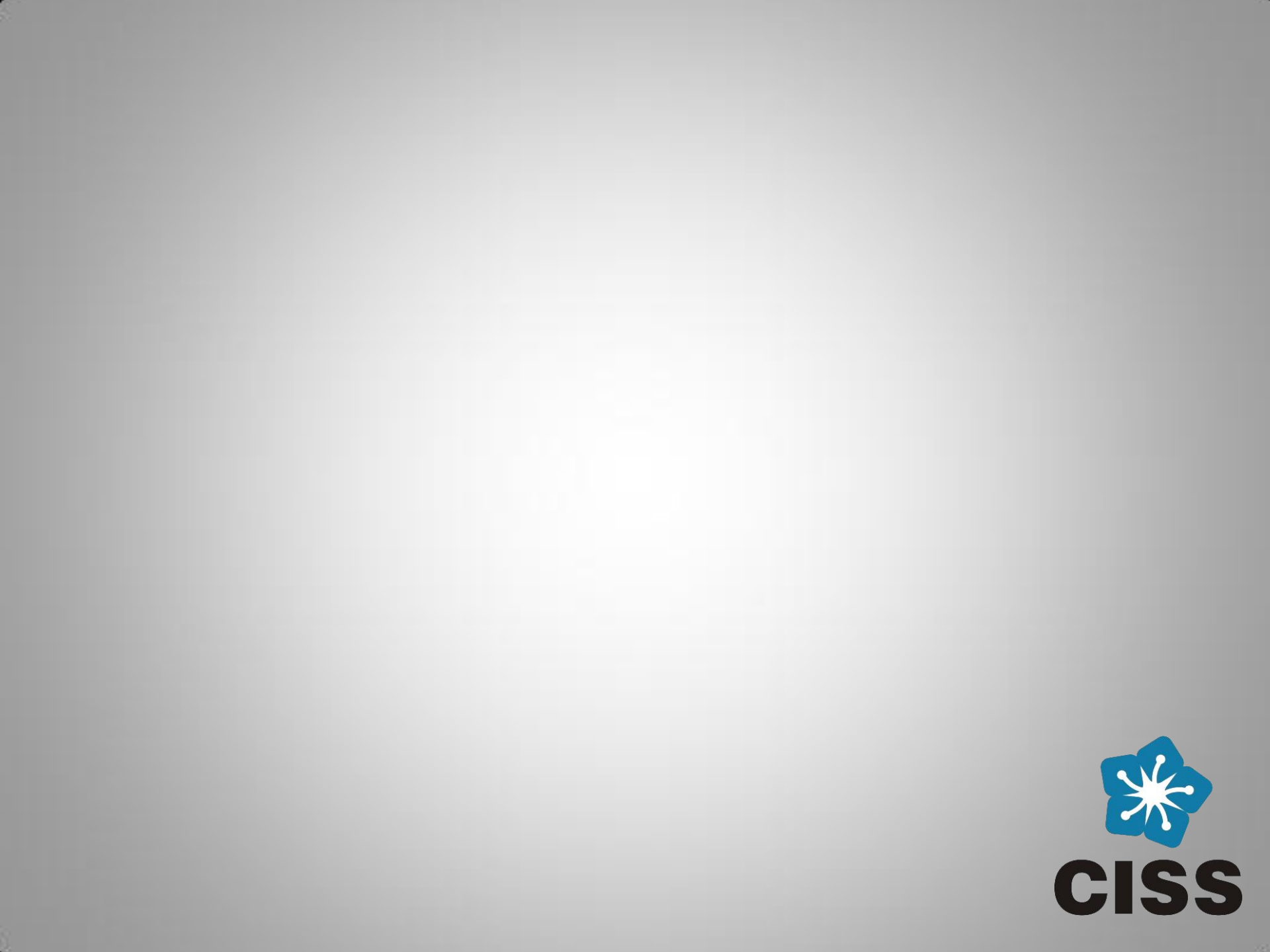
- Контакты:
- Twitter.com/NTarakanov
- Nikita.Tarakanov.Researcher@gmail.com

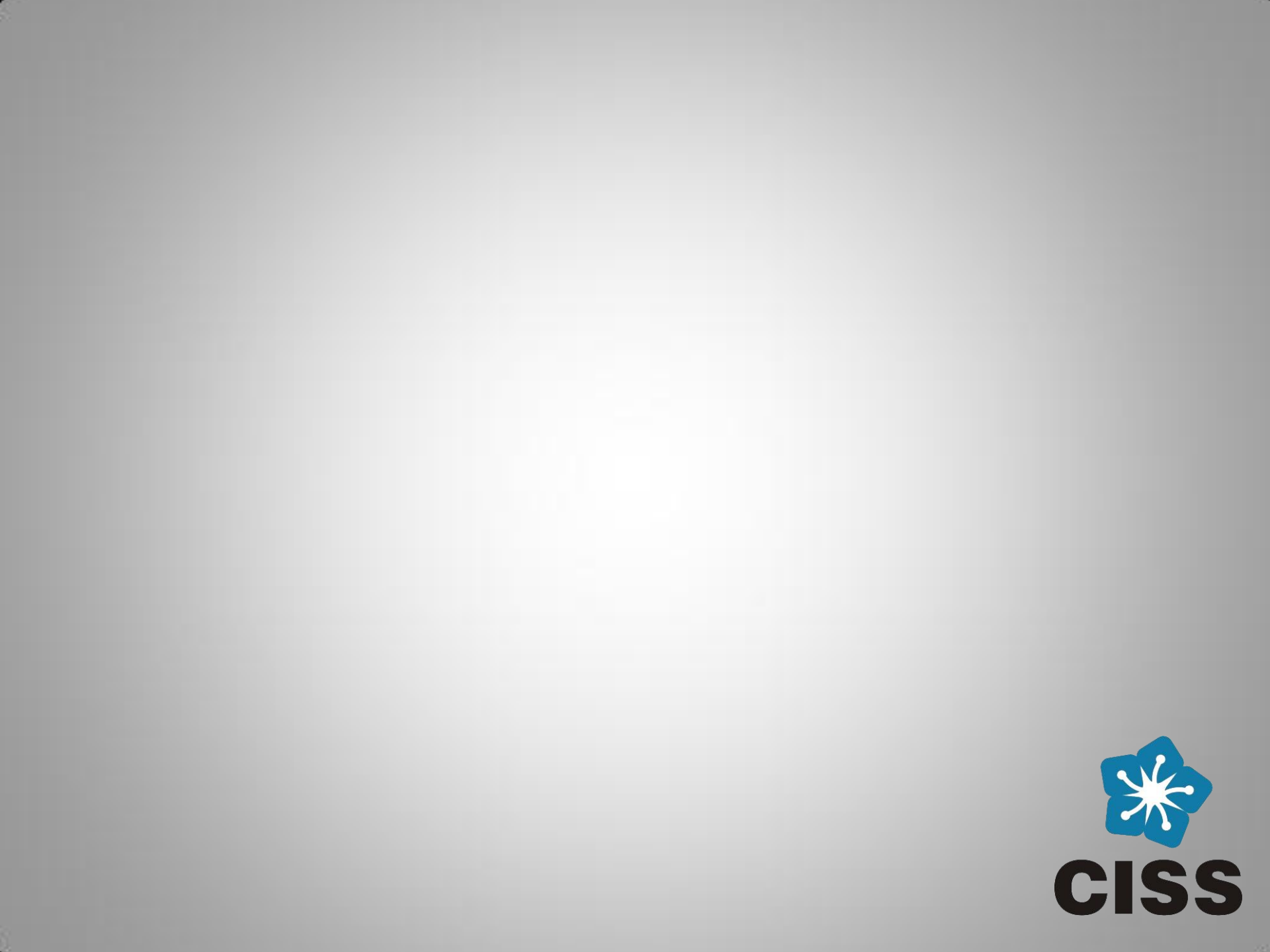
References

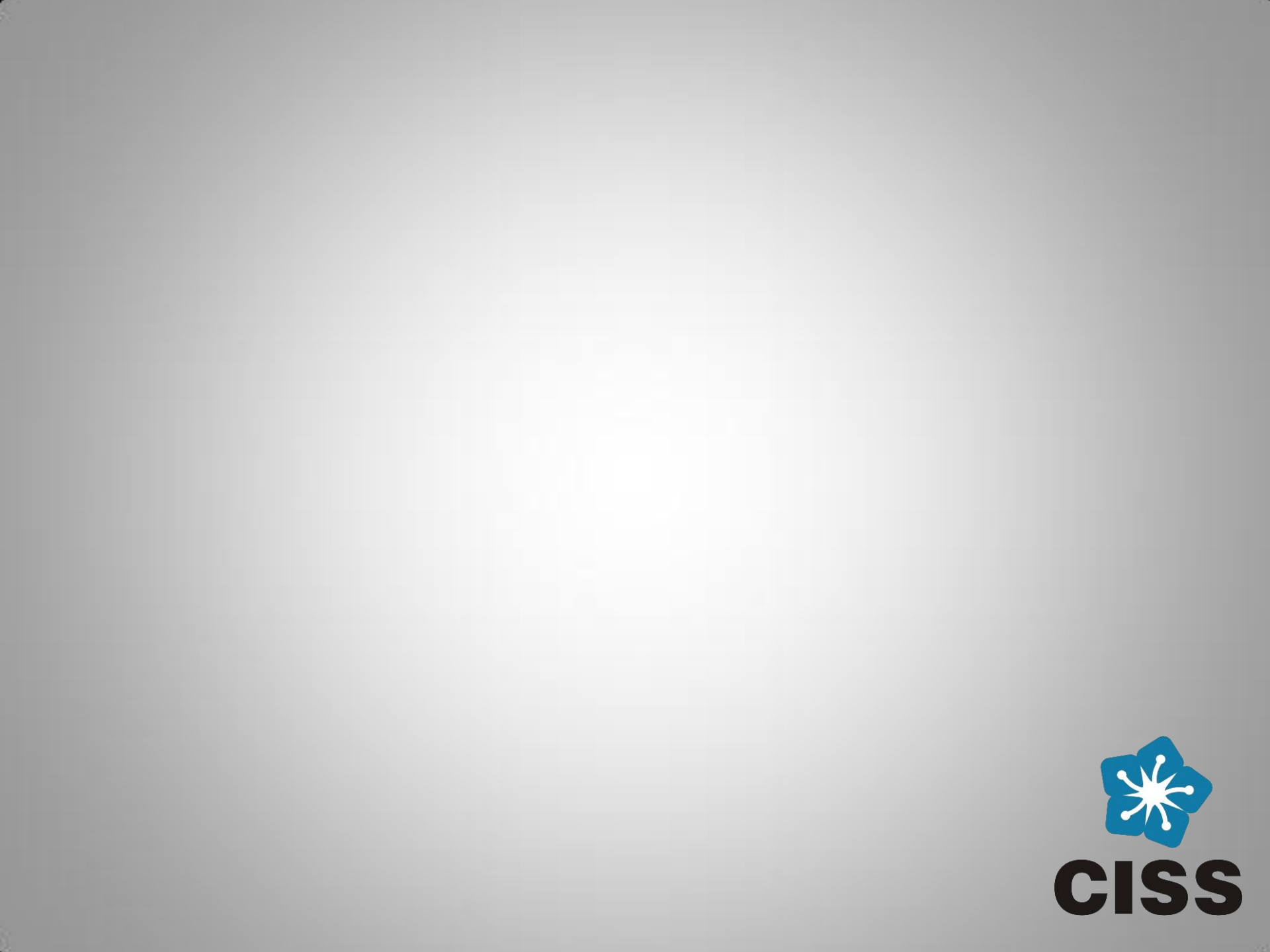
- Kortchinsky[2008] – Kostya Kortchinsky
- Real-World Kernel Pool Exploitation,
- SyScan 2008 Hong Kong
- Tarjei Mandt[2010] – Kernel Pool Explotation at Infiltrate

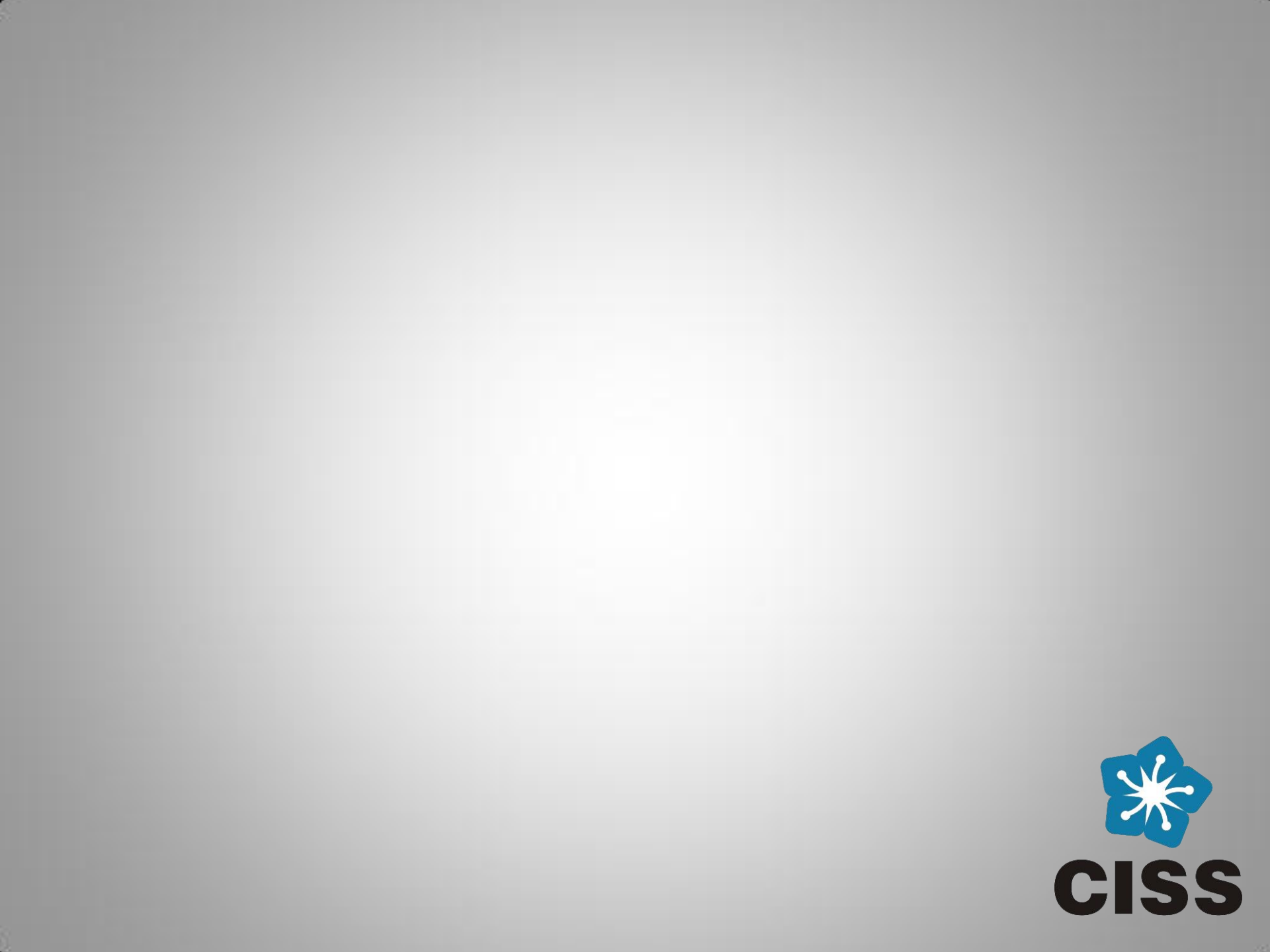

CISS

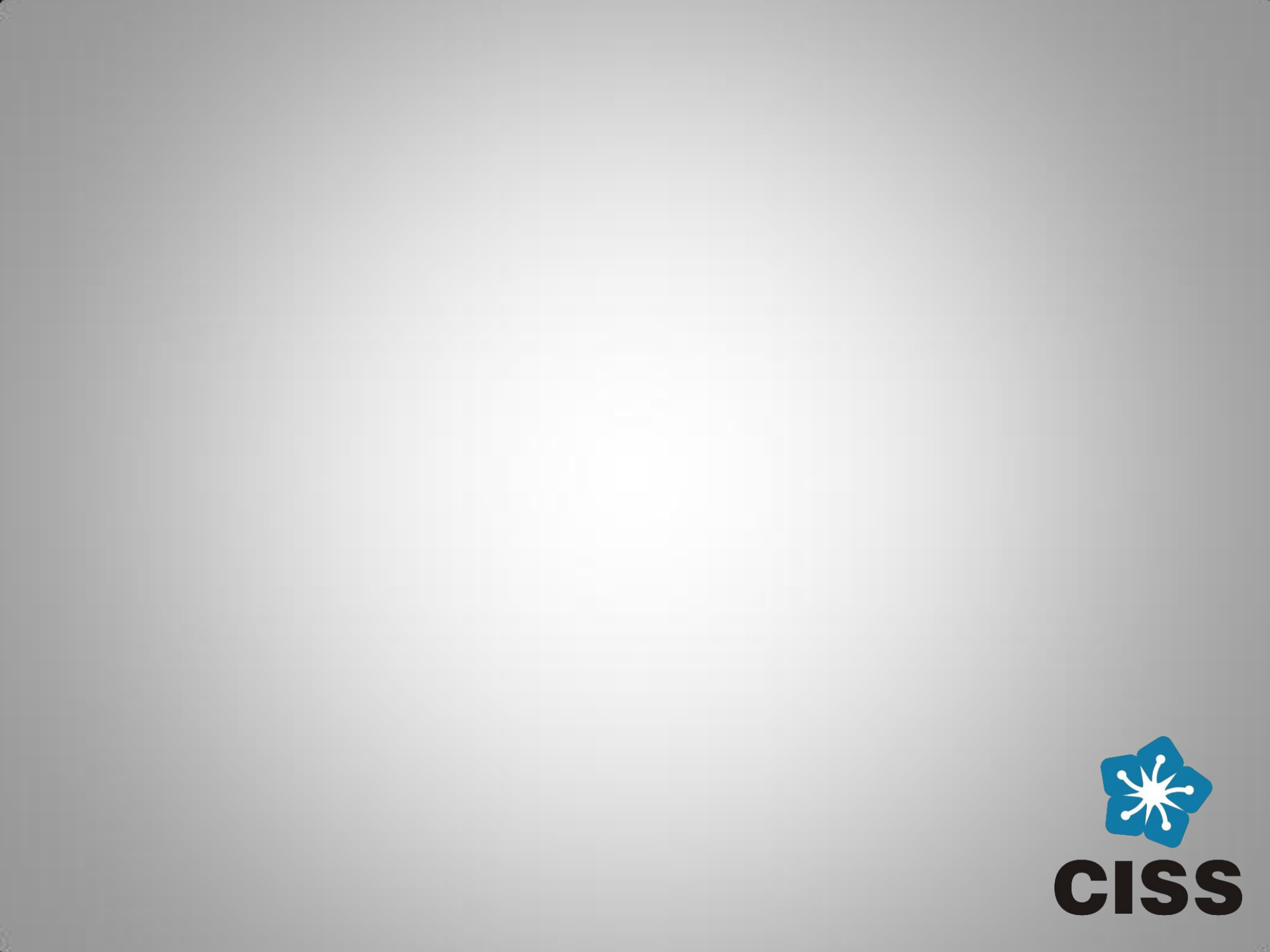

CISS

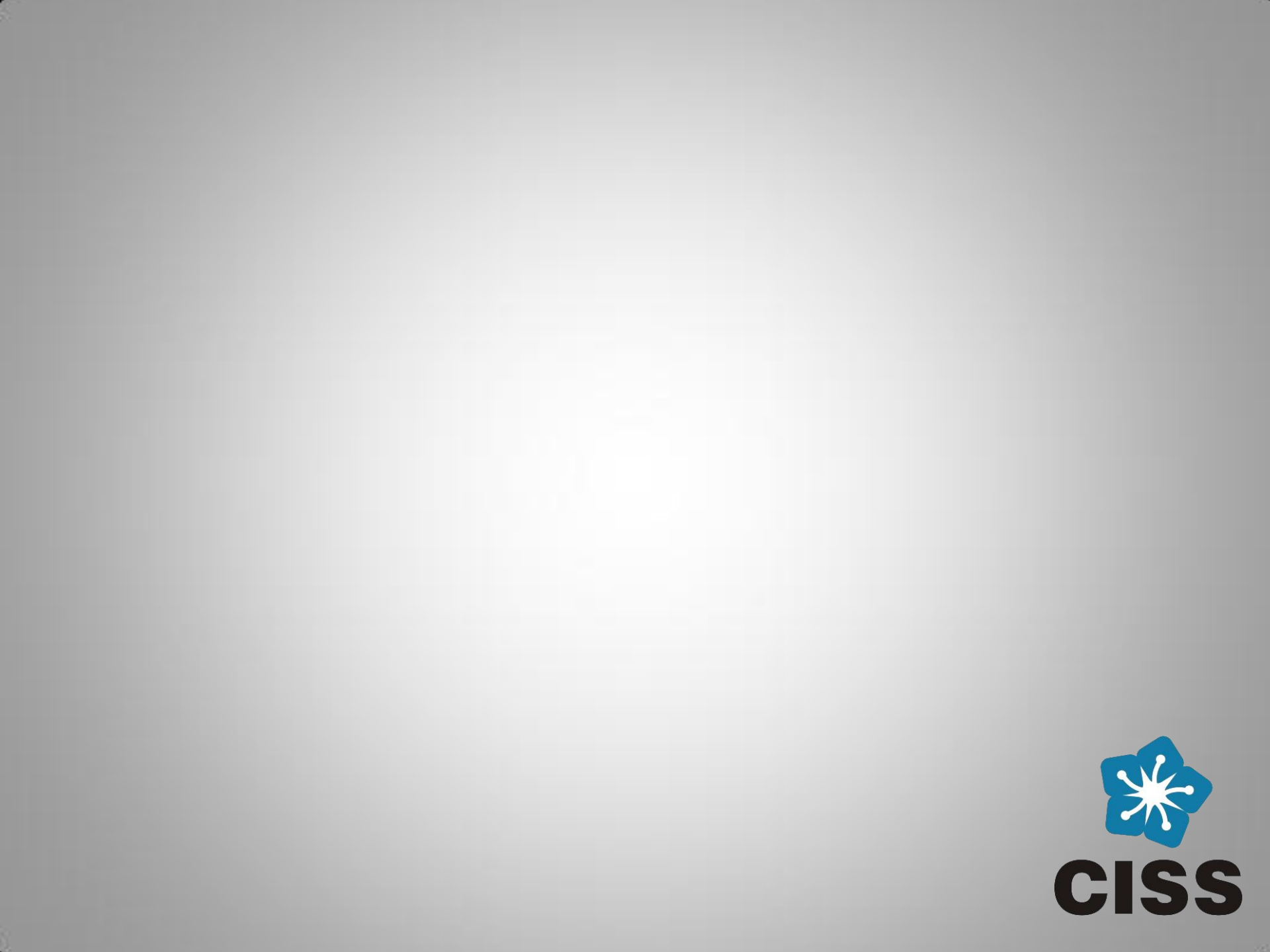

CISS

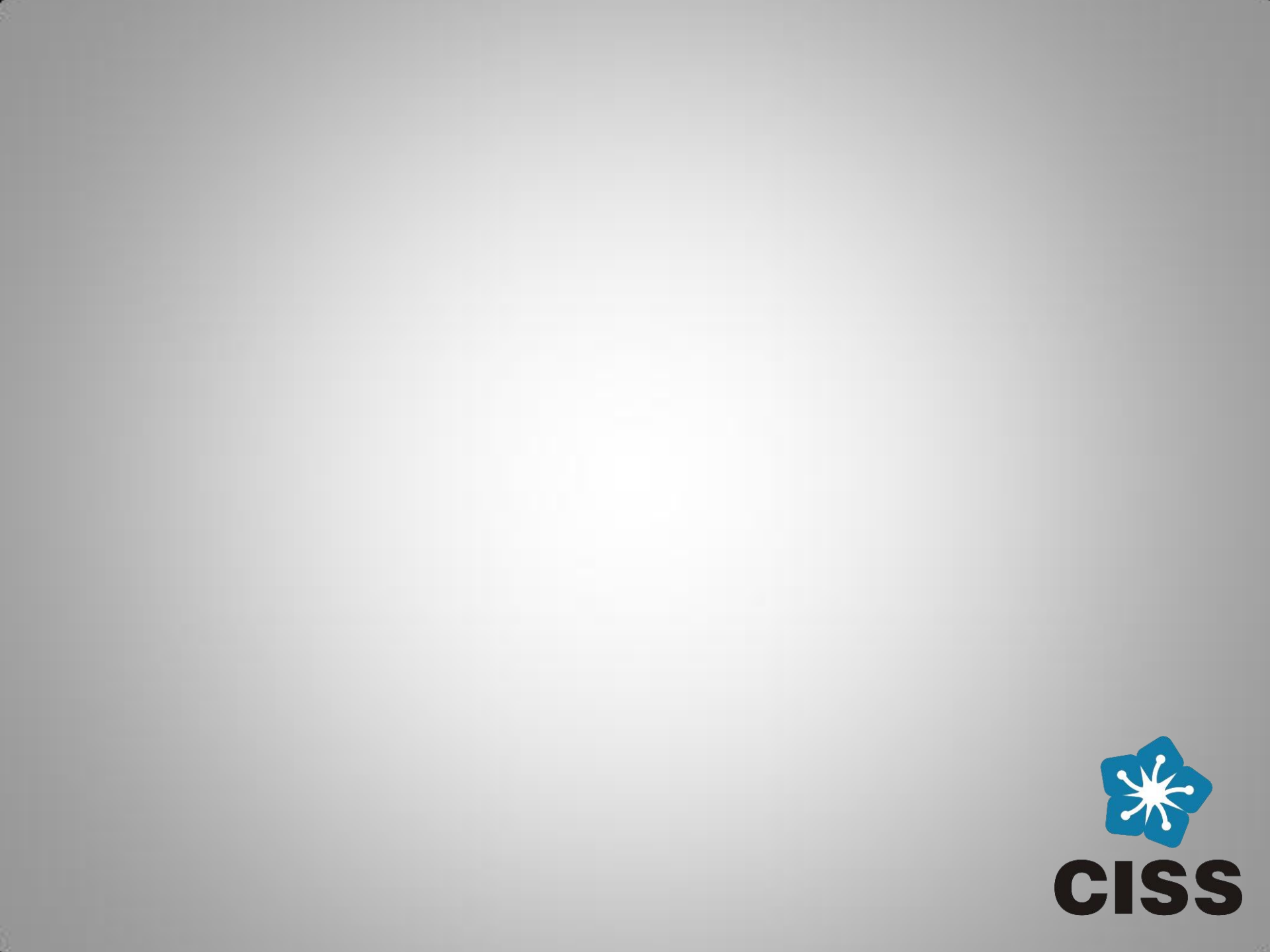

CISS

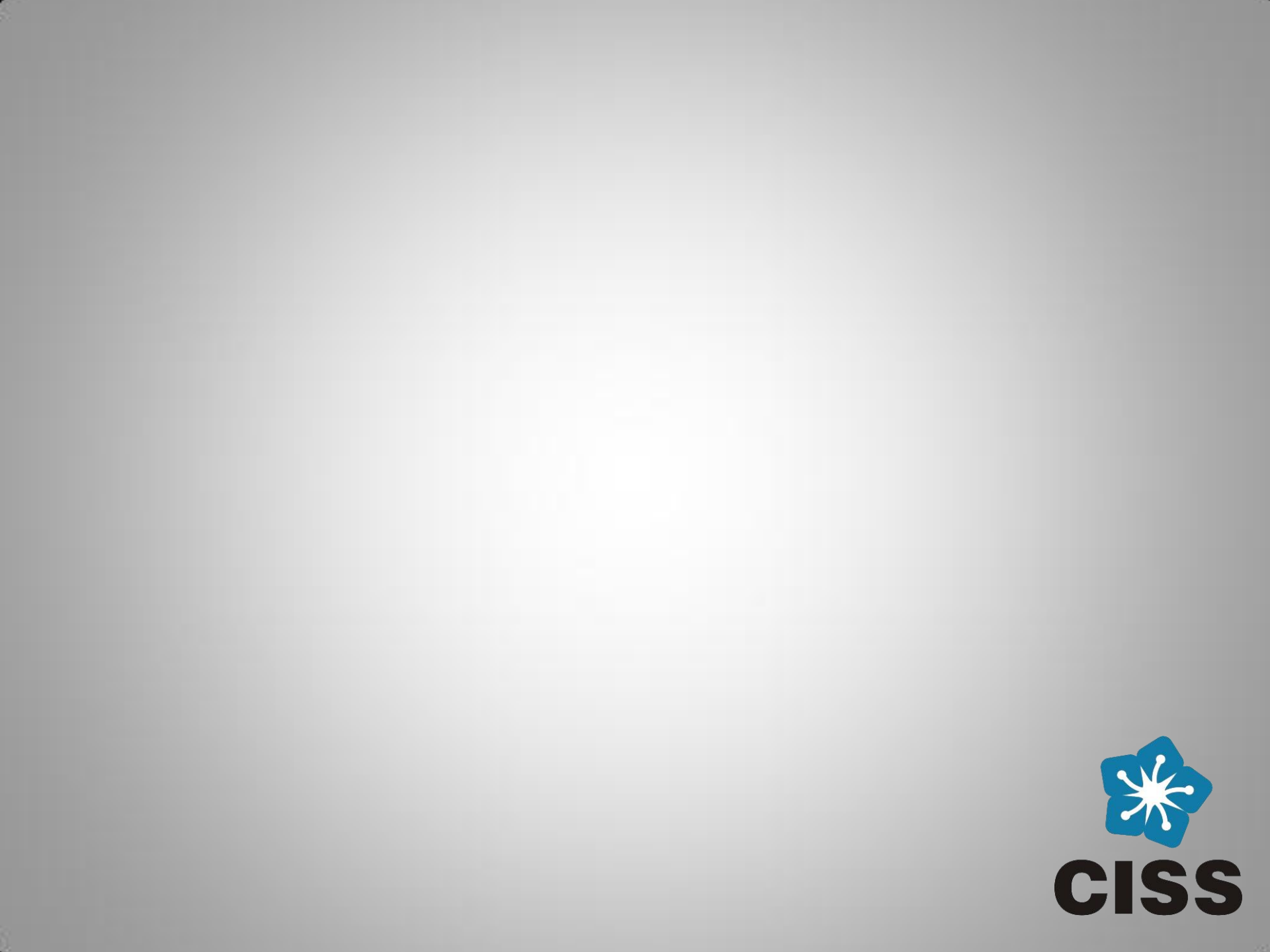

CISS

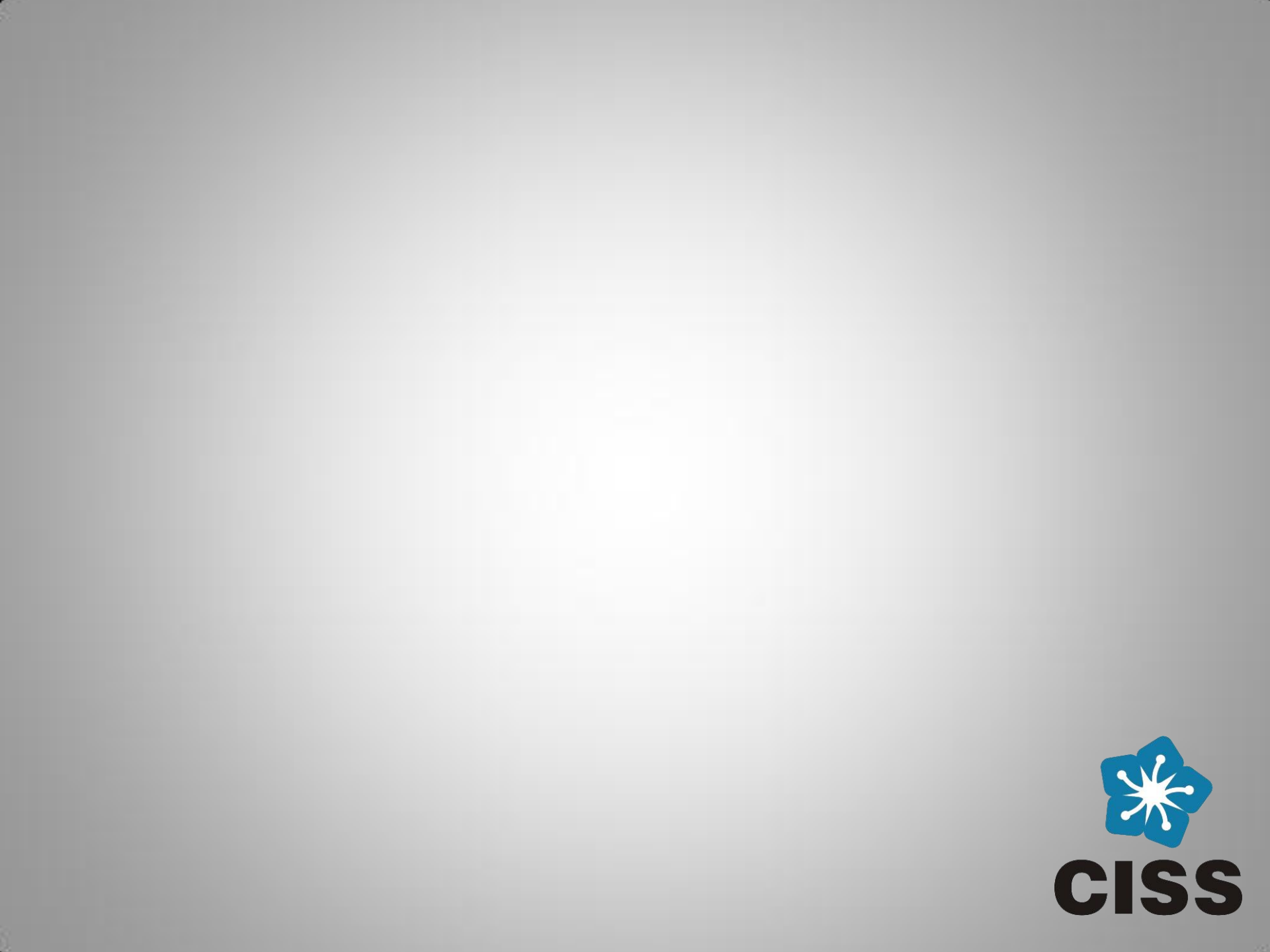

CISS

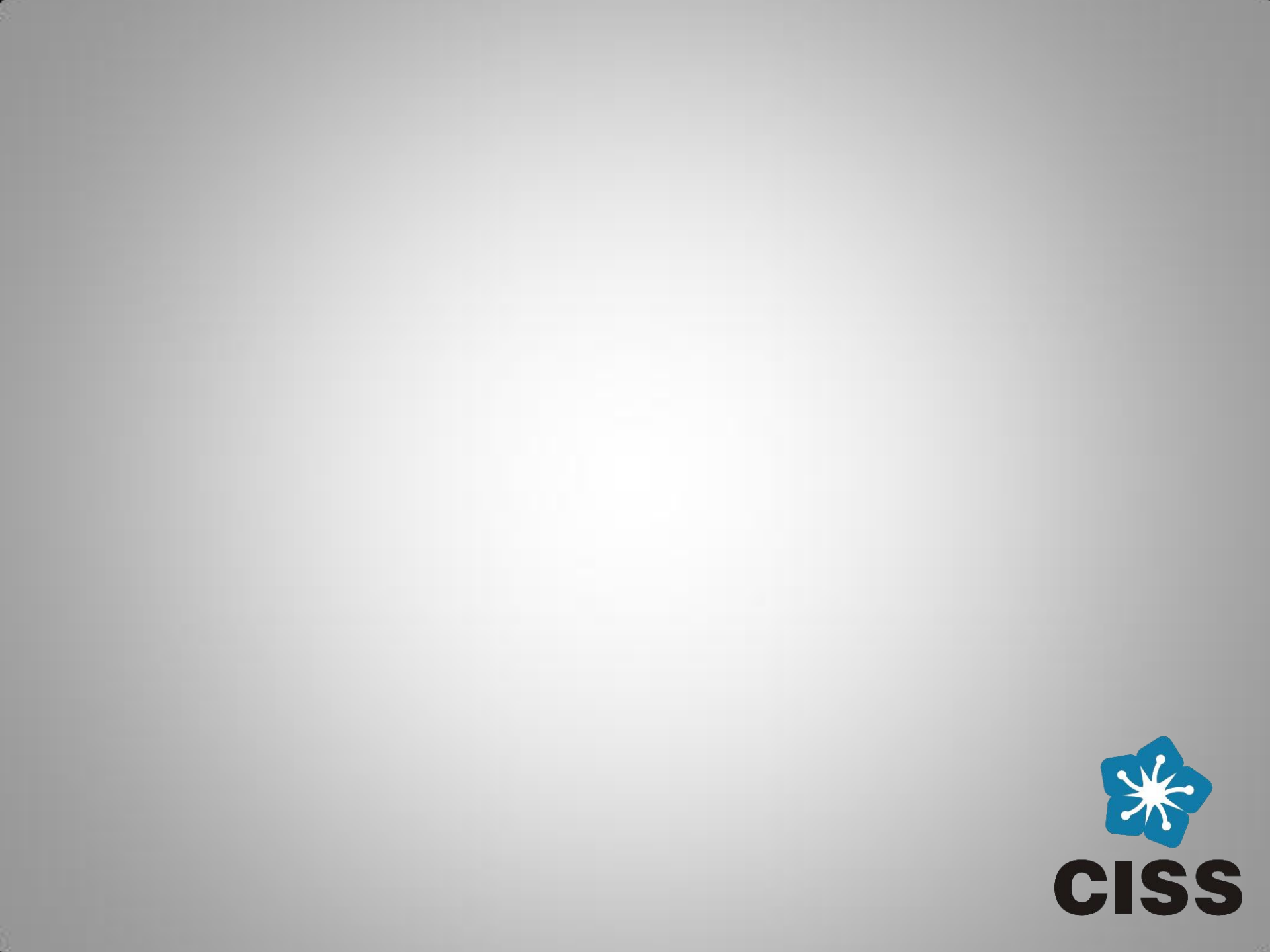

CISS

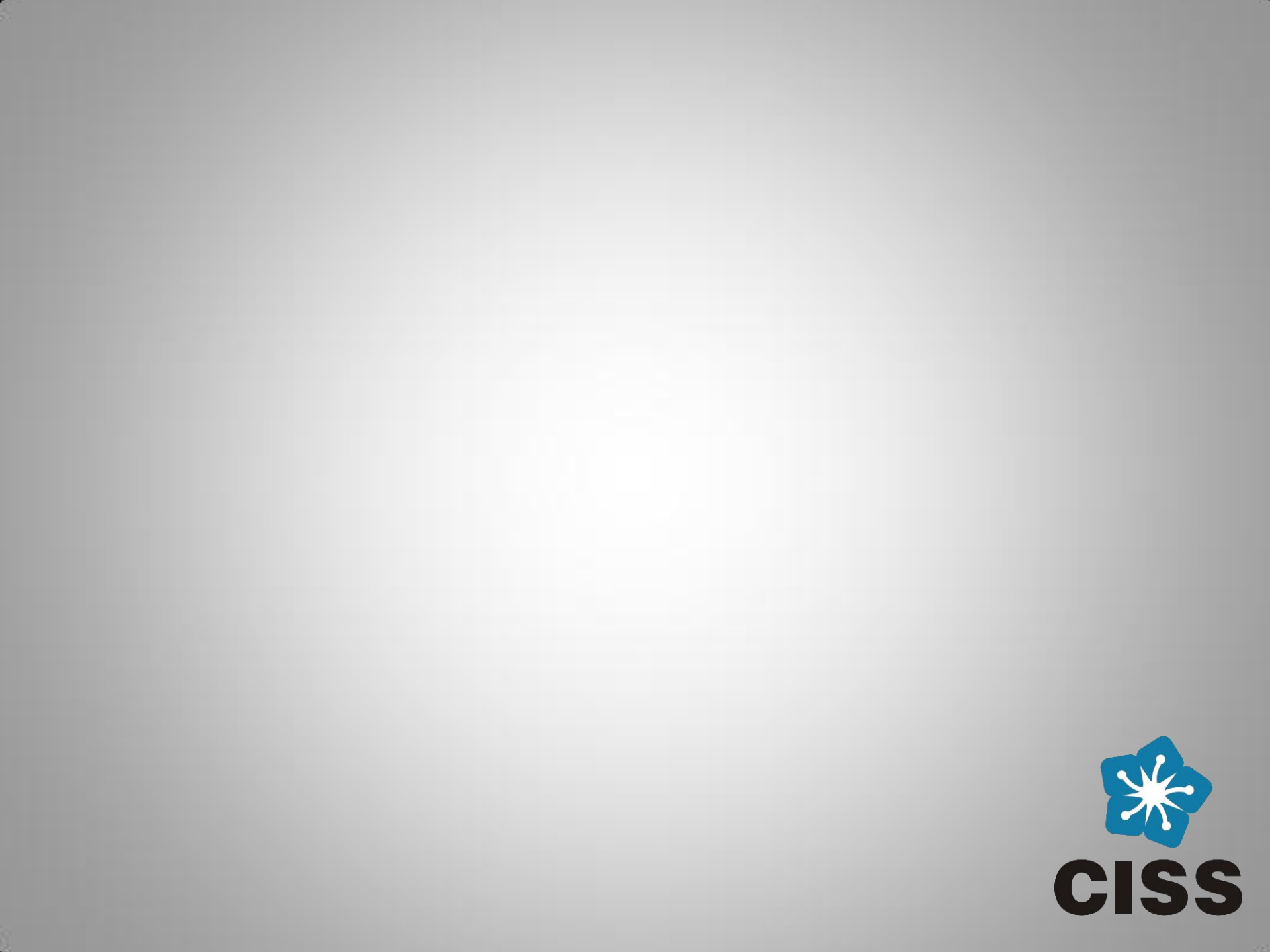

CISS

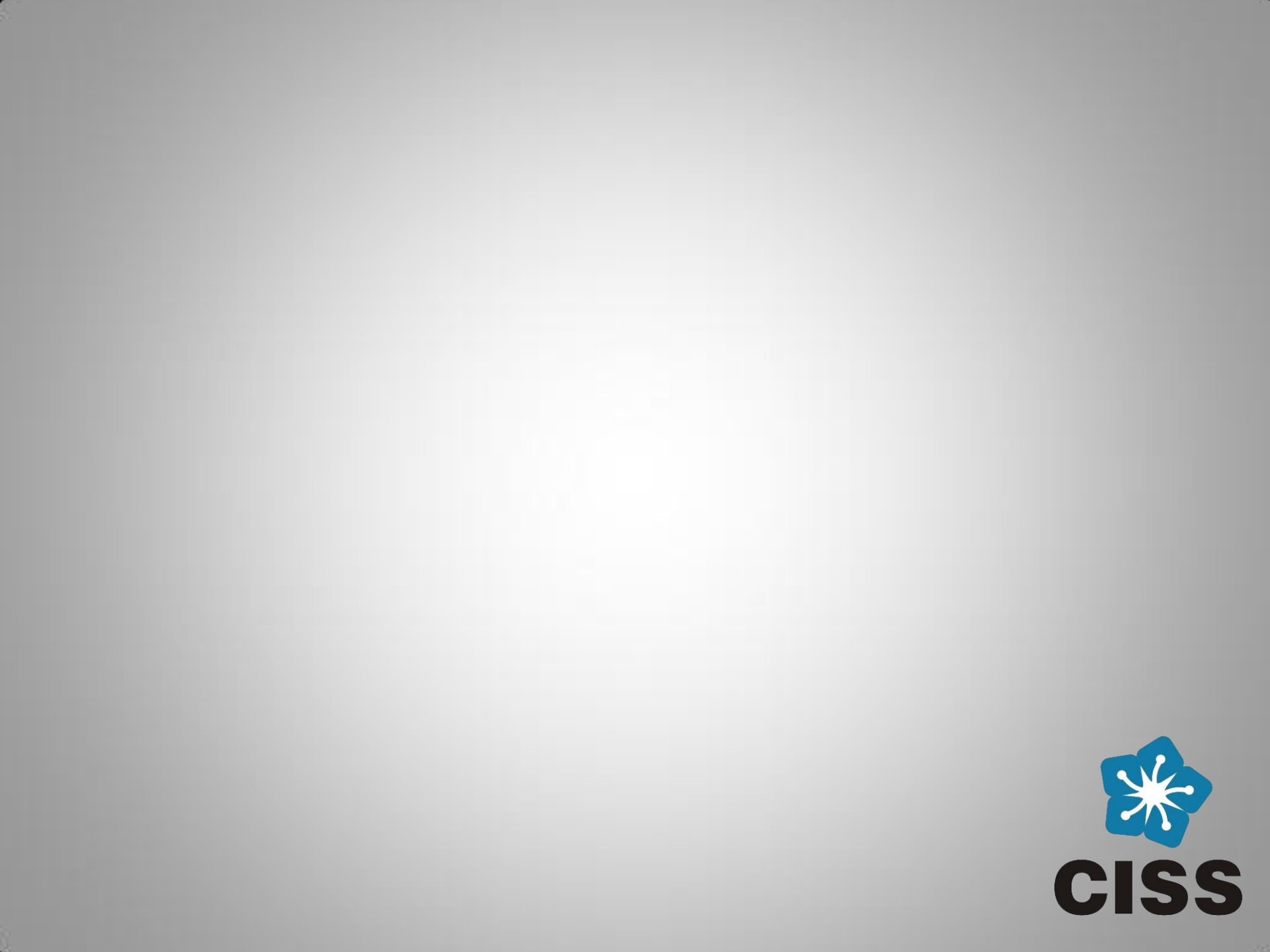

CISS

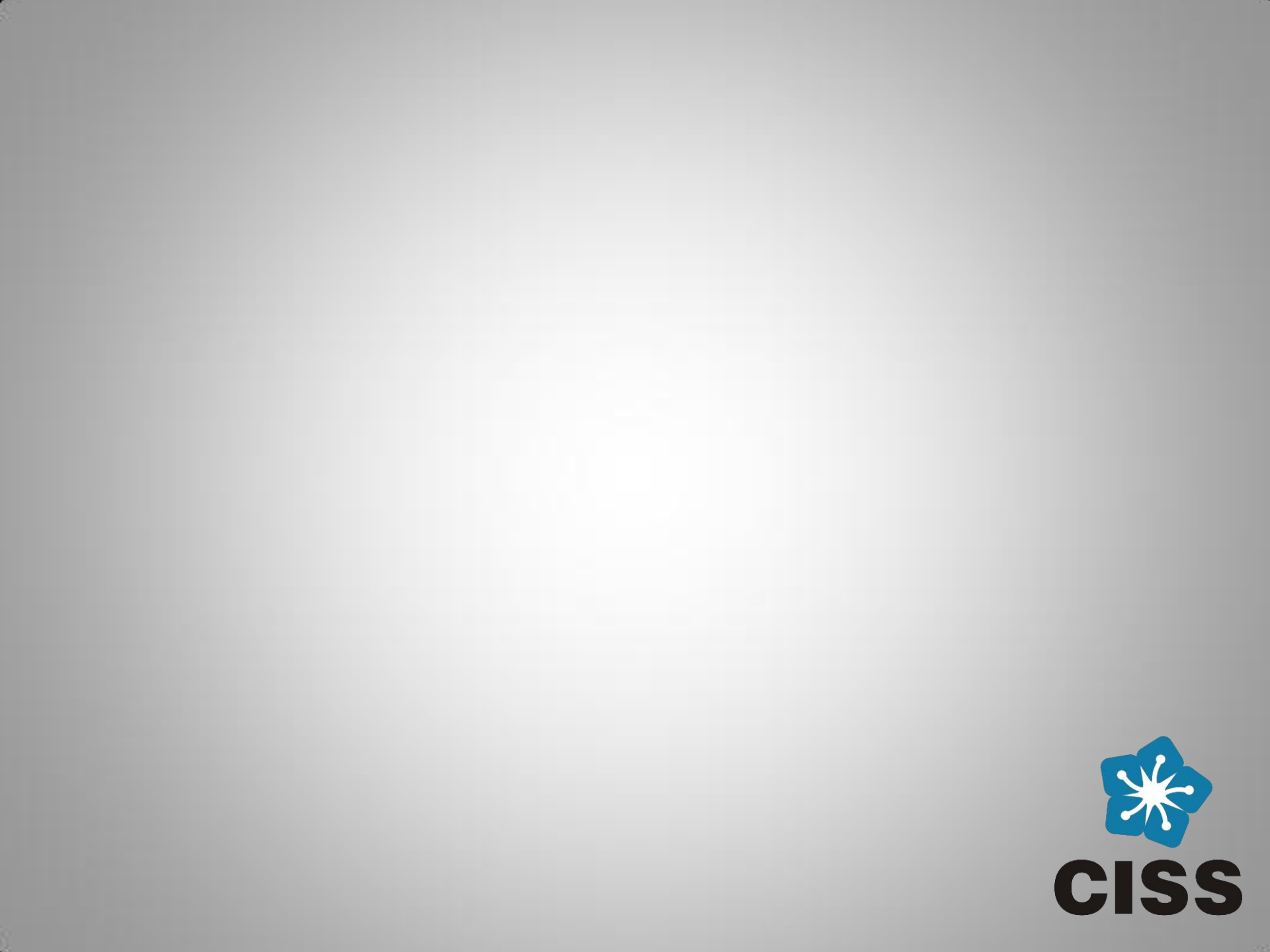

CISS

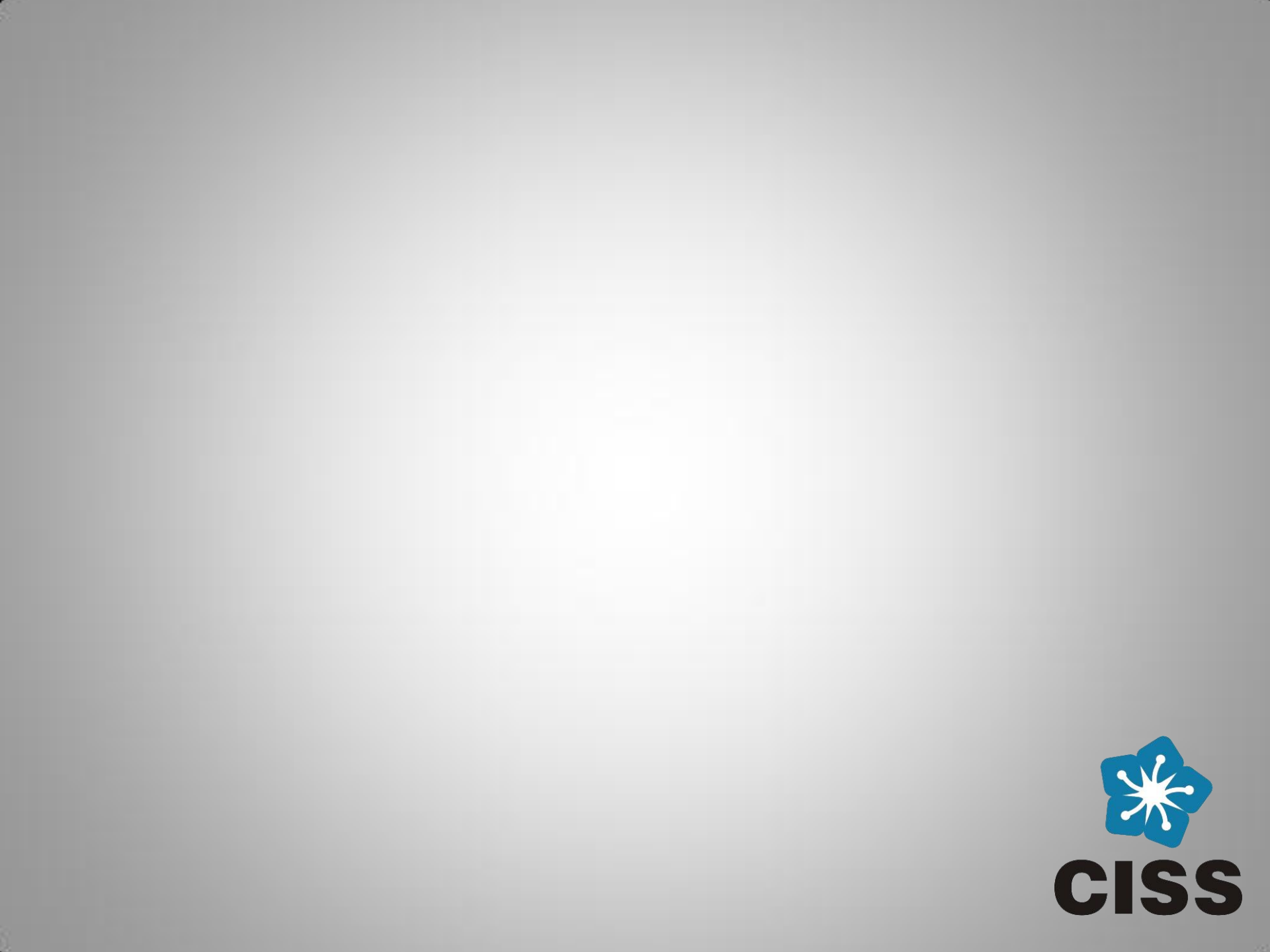

CISS

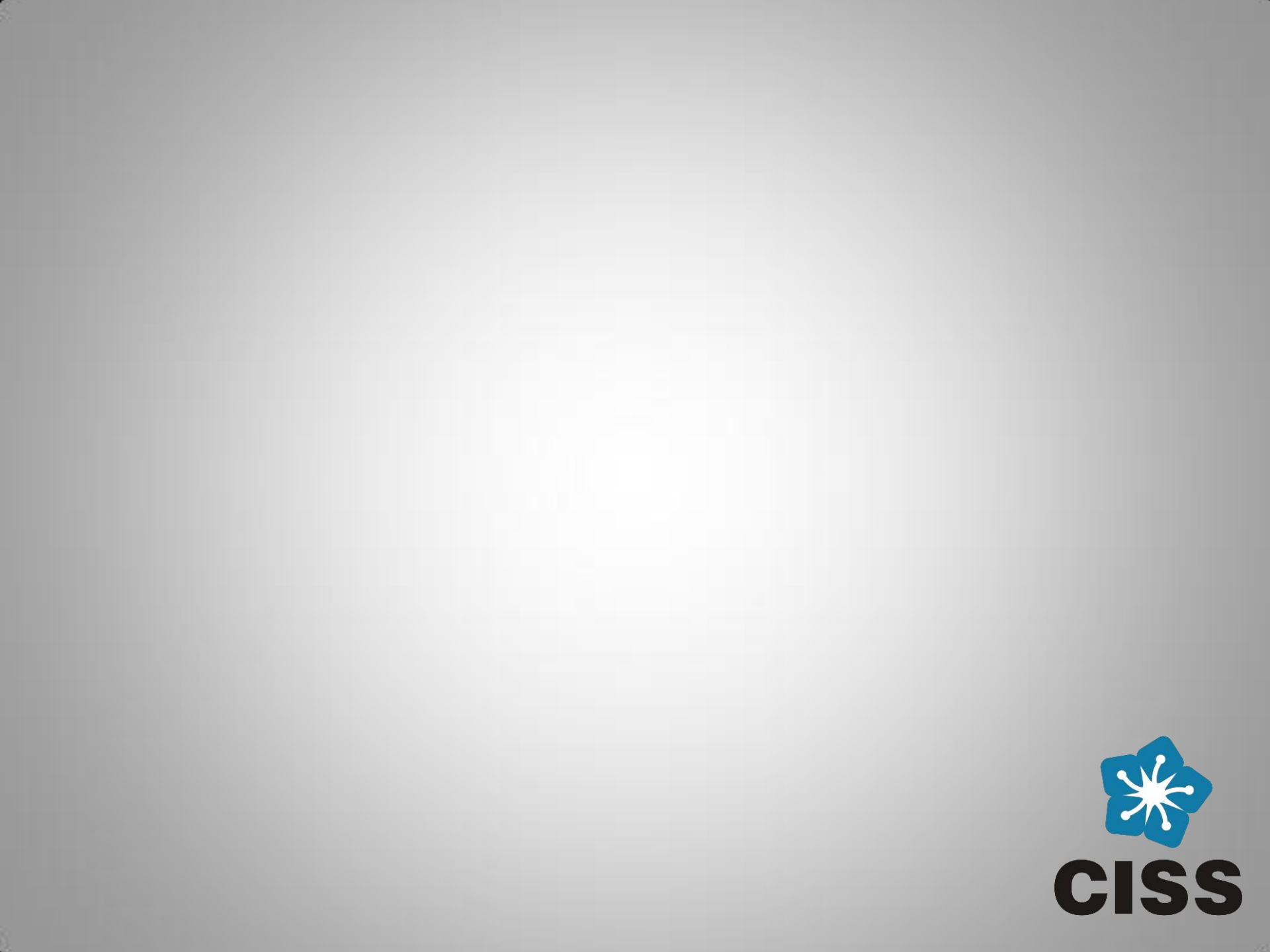

CISS

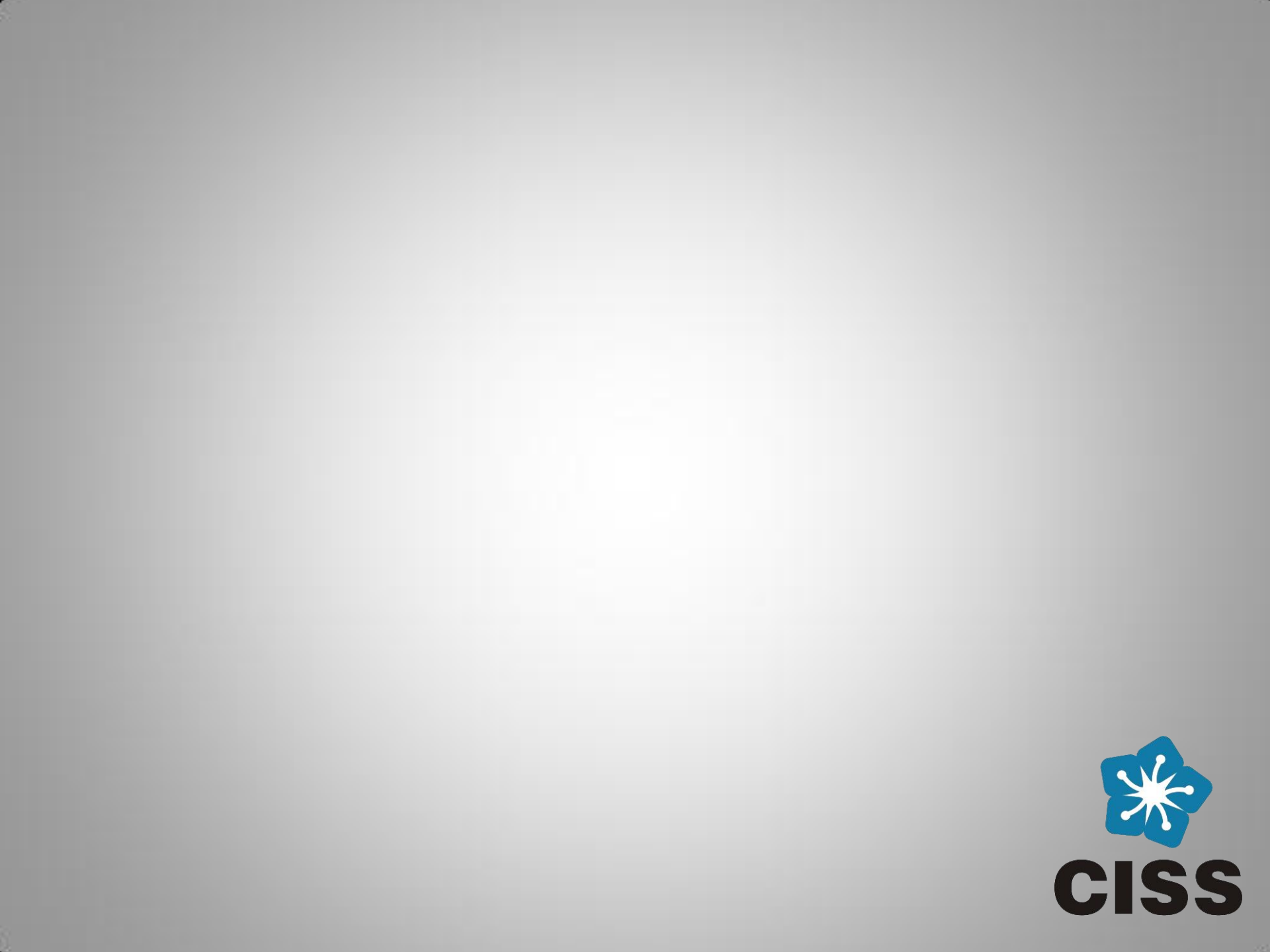

CISS

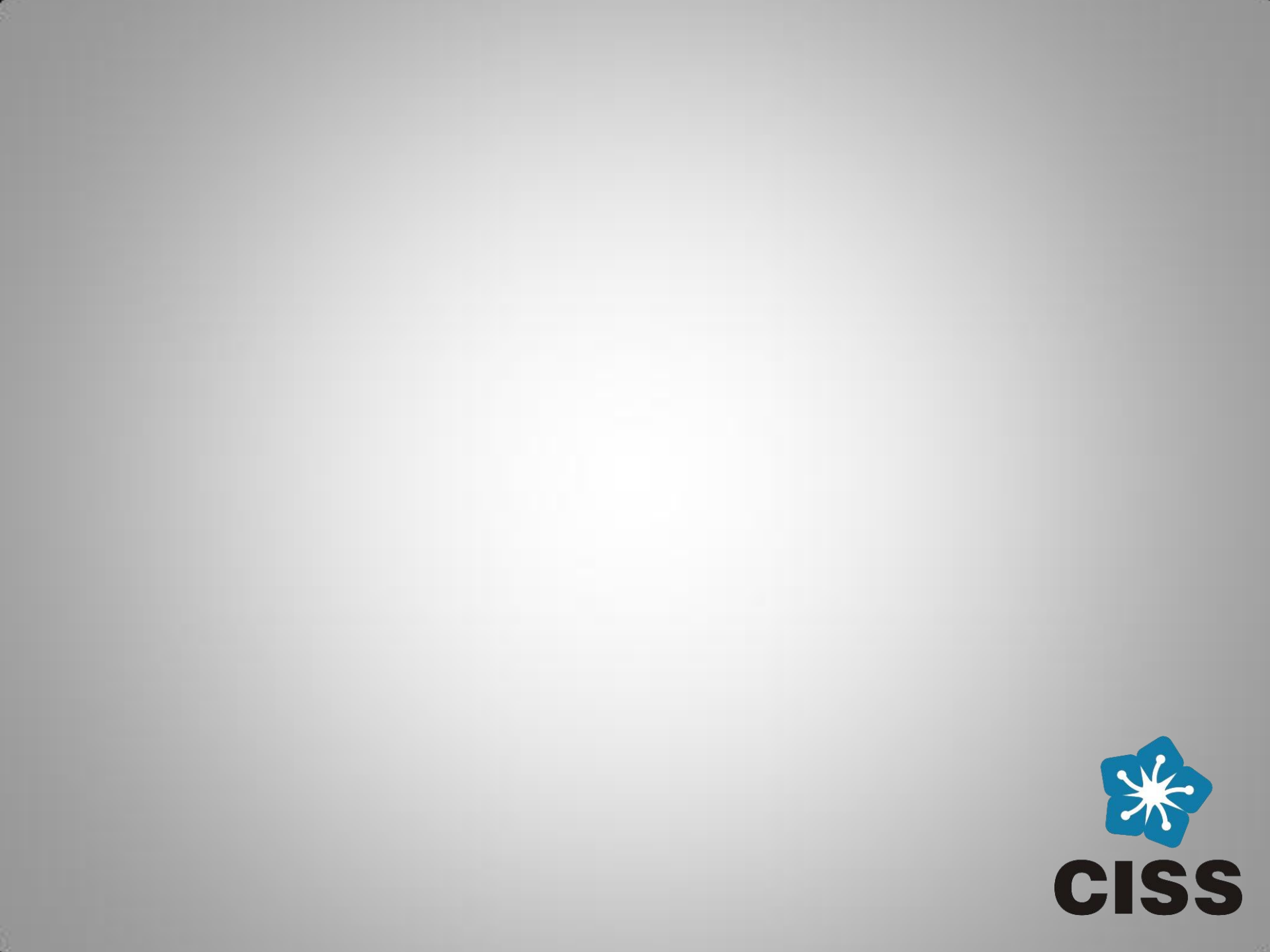

CISS

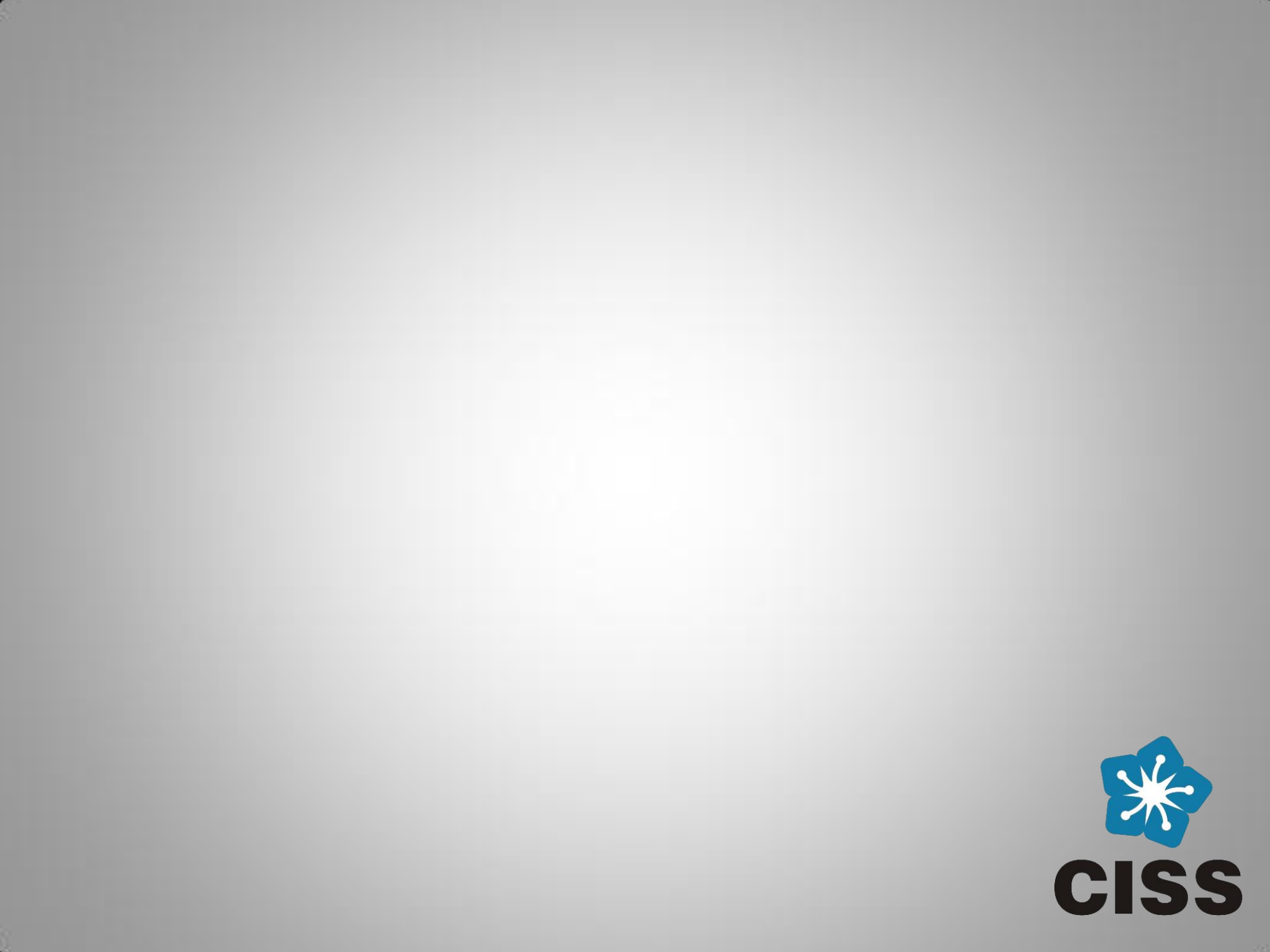

CISS

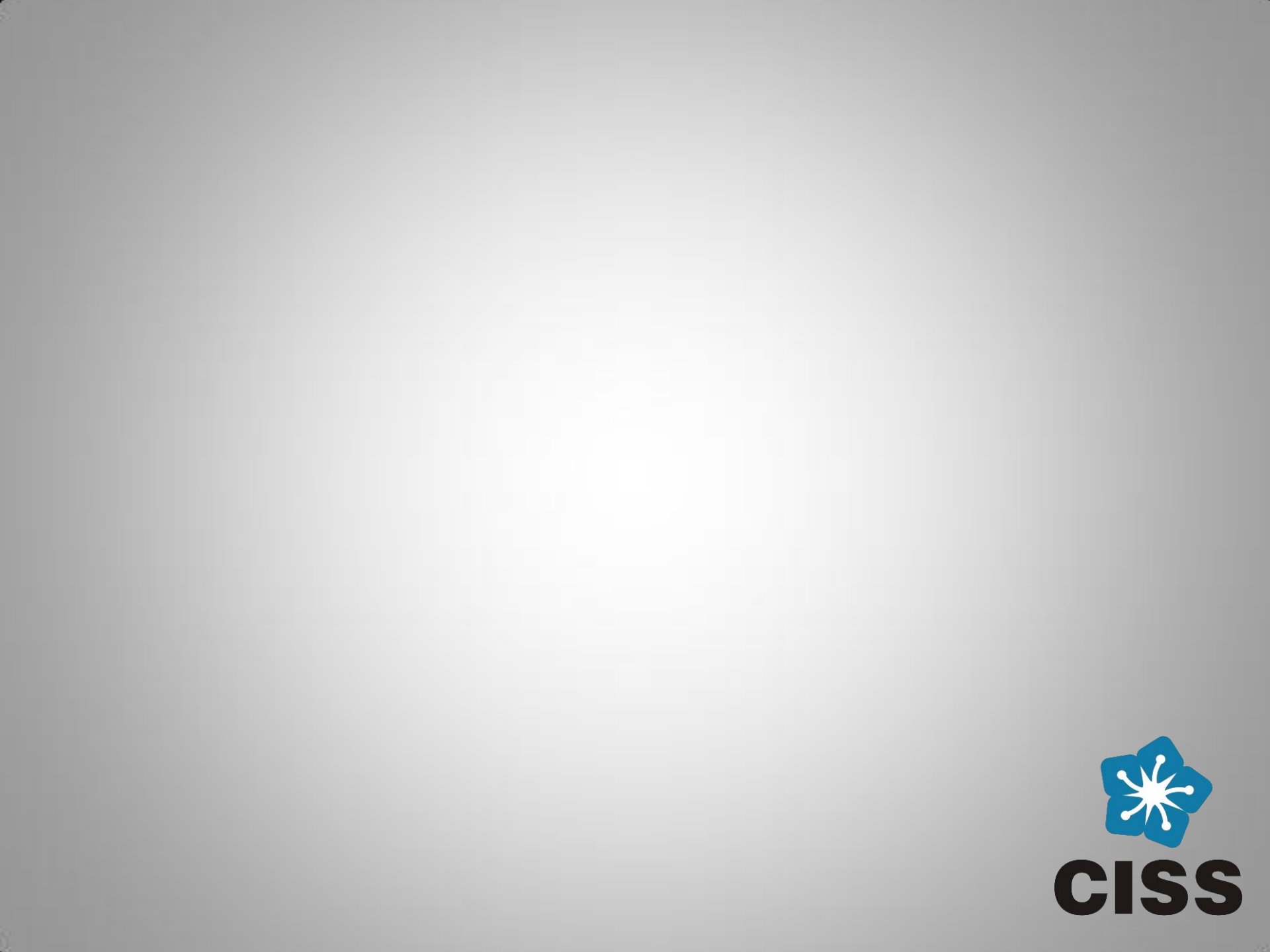

CISS

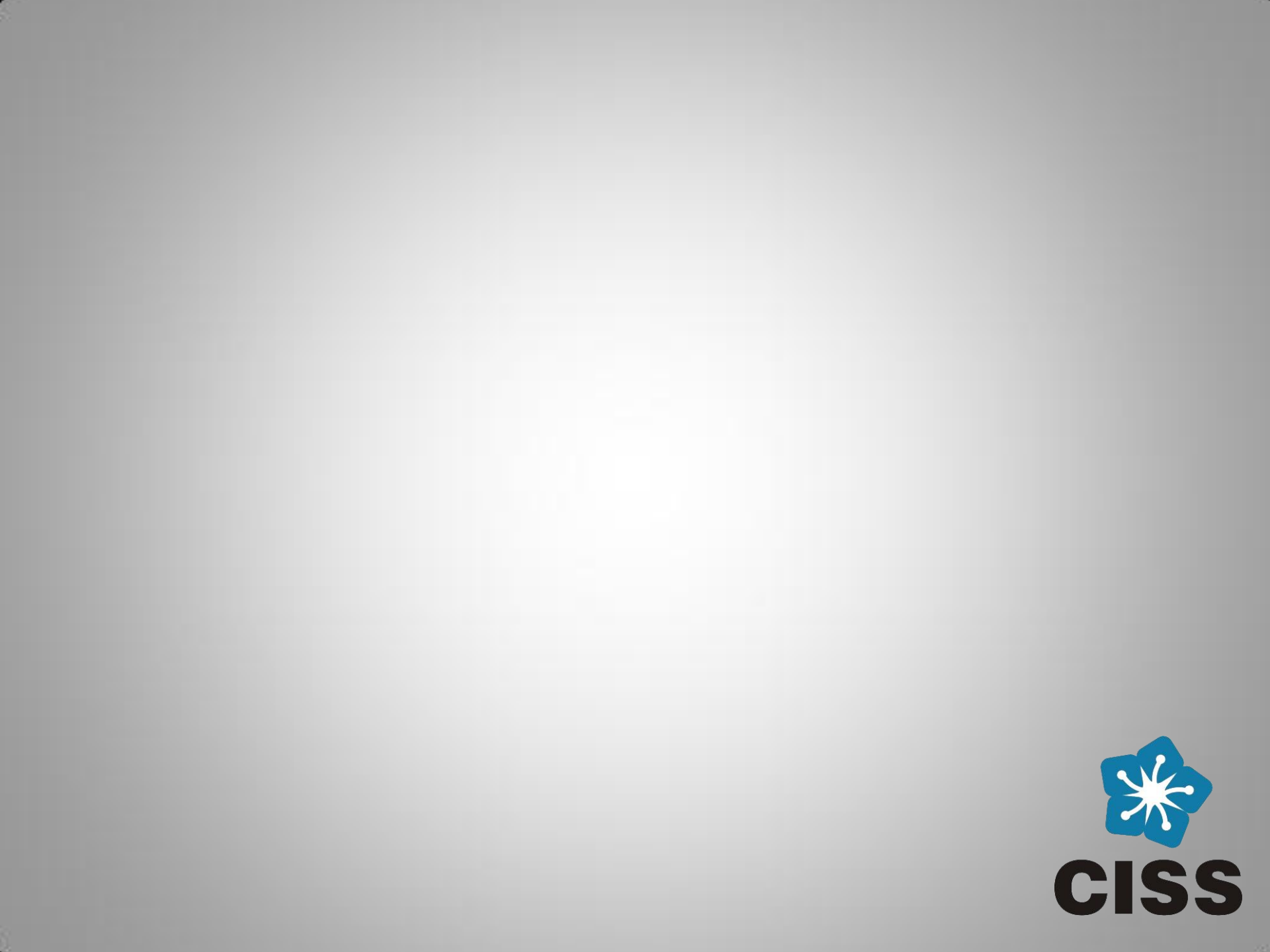

CISS

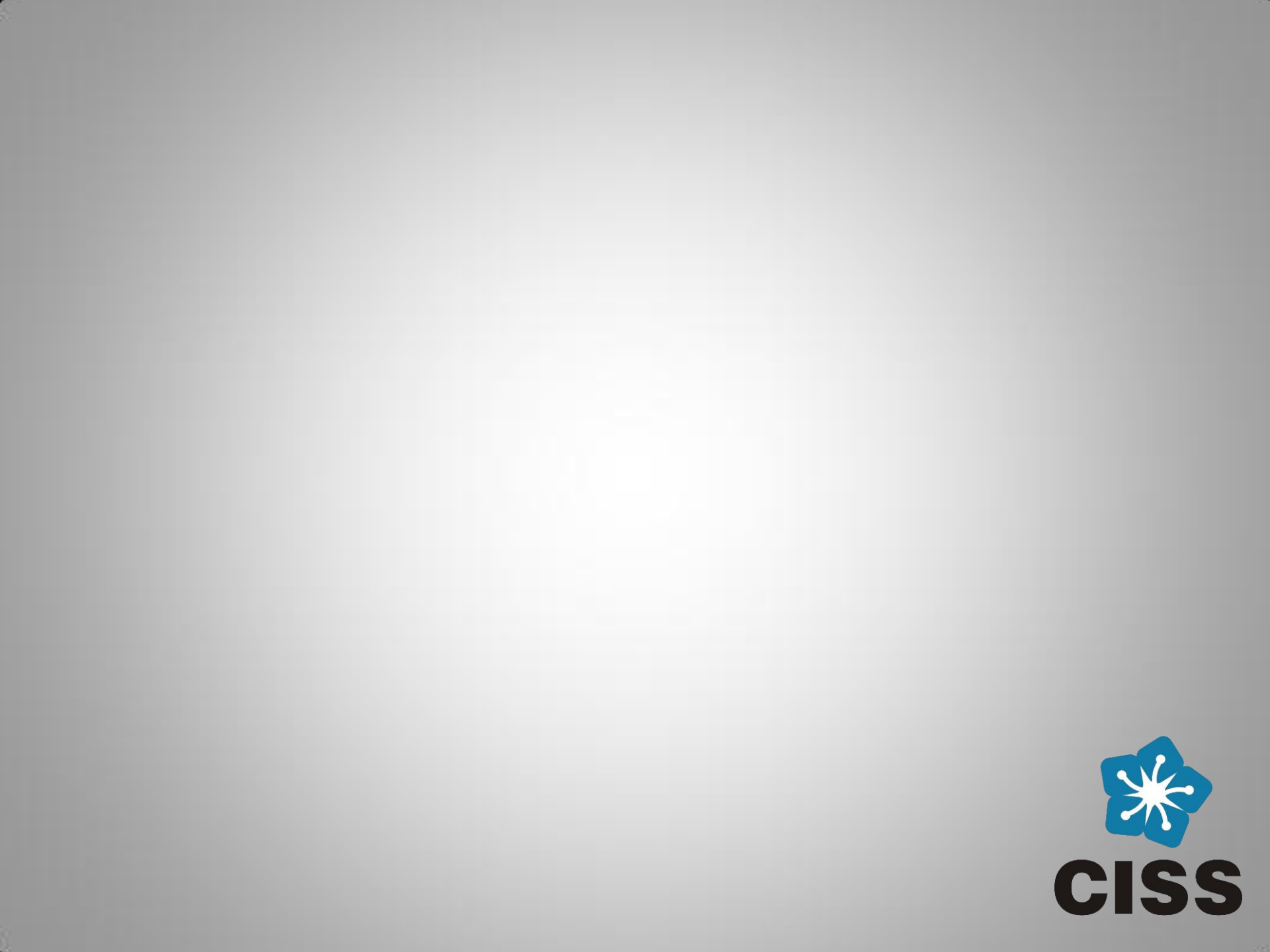

CISS

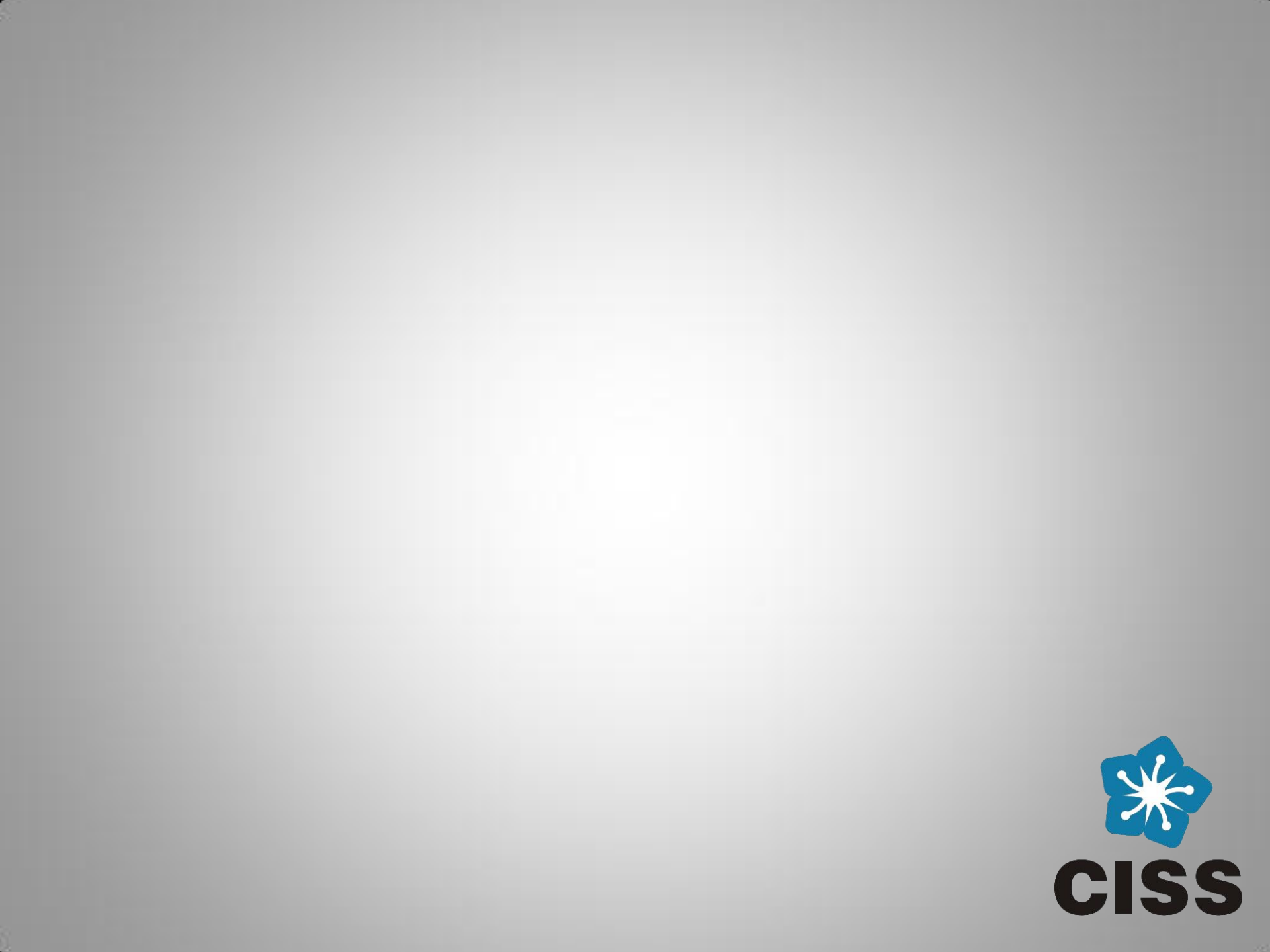

CISS

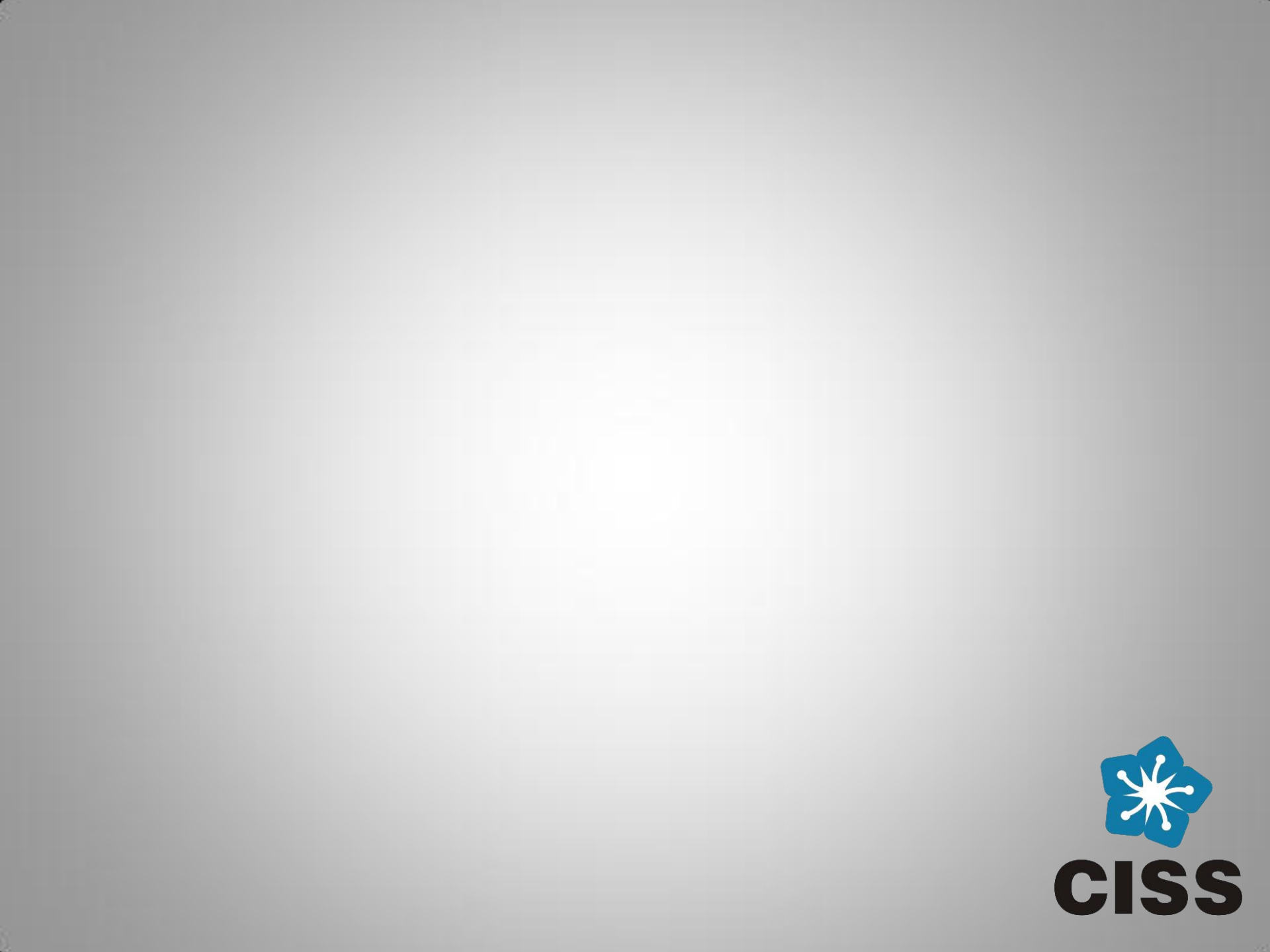

CISS

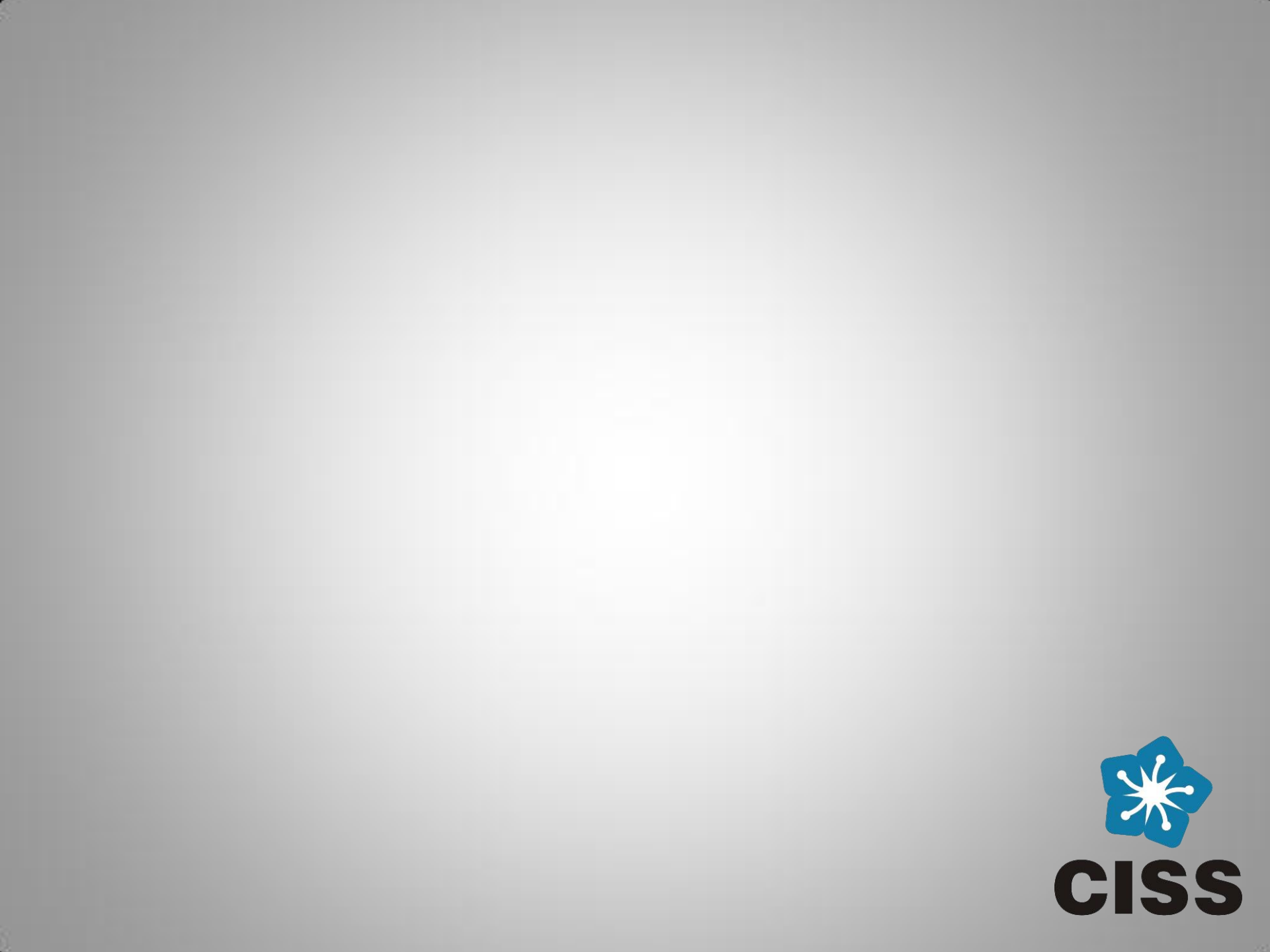

CISS

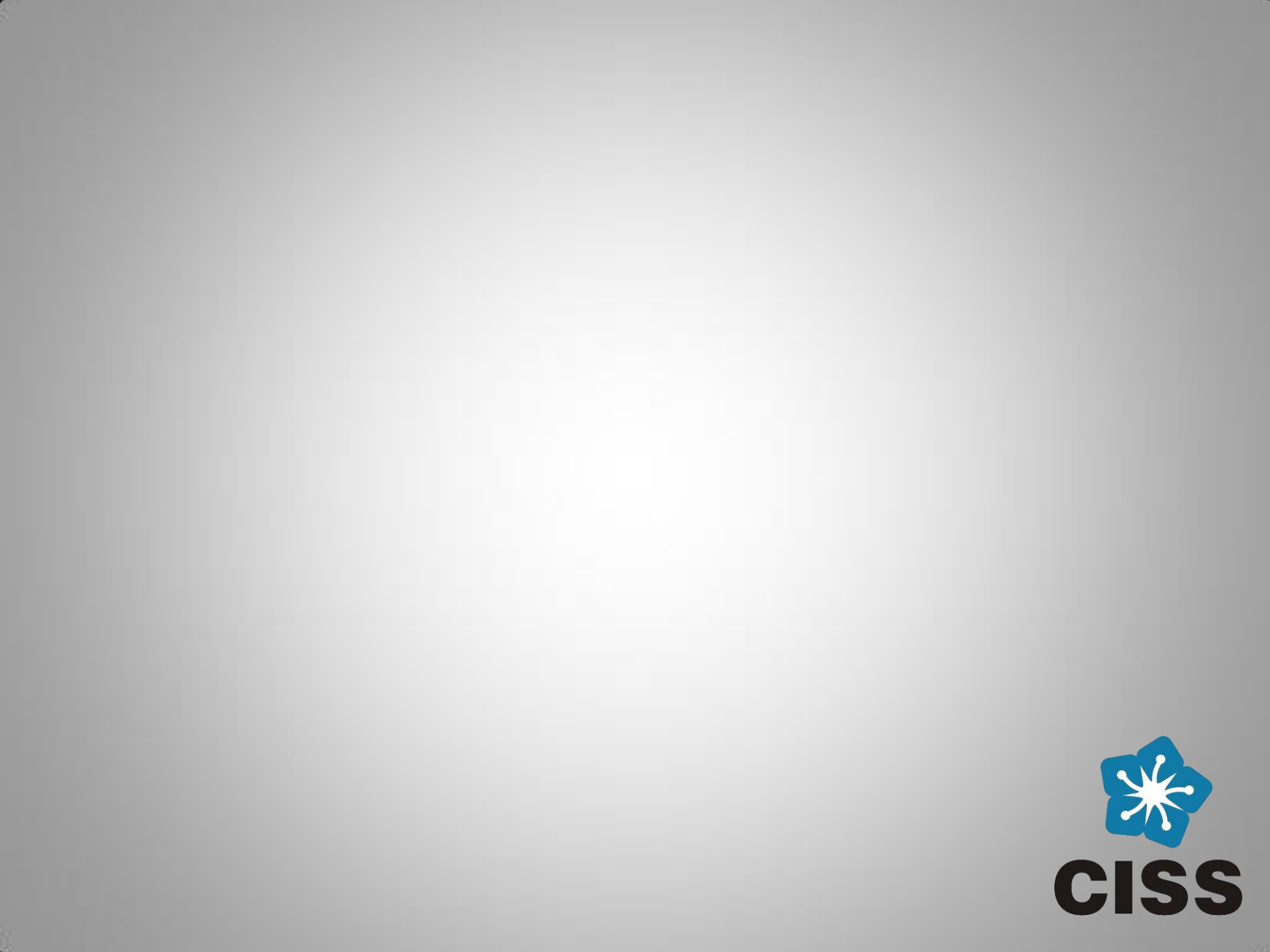

CISS

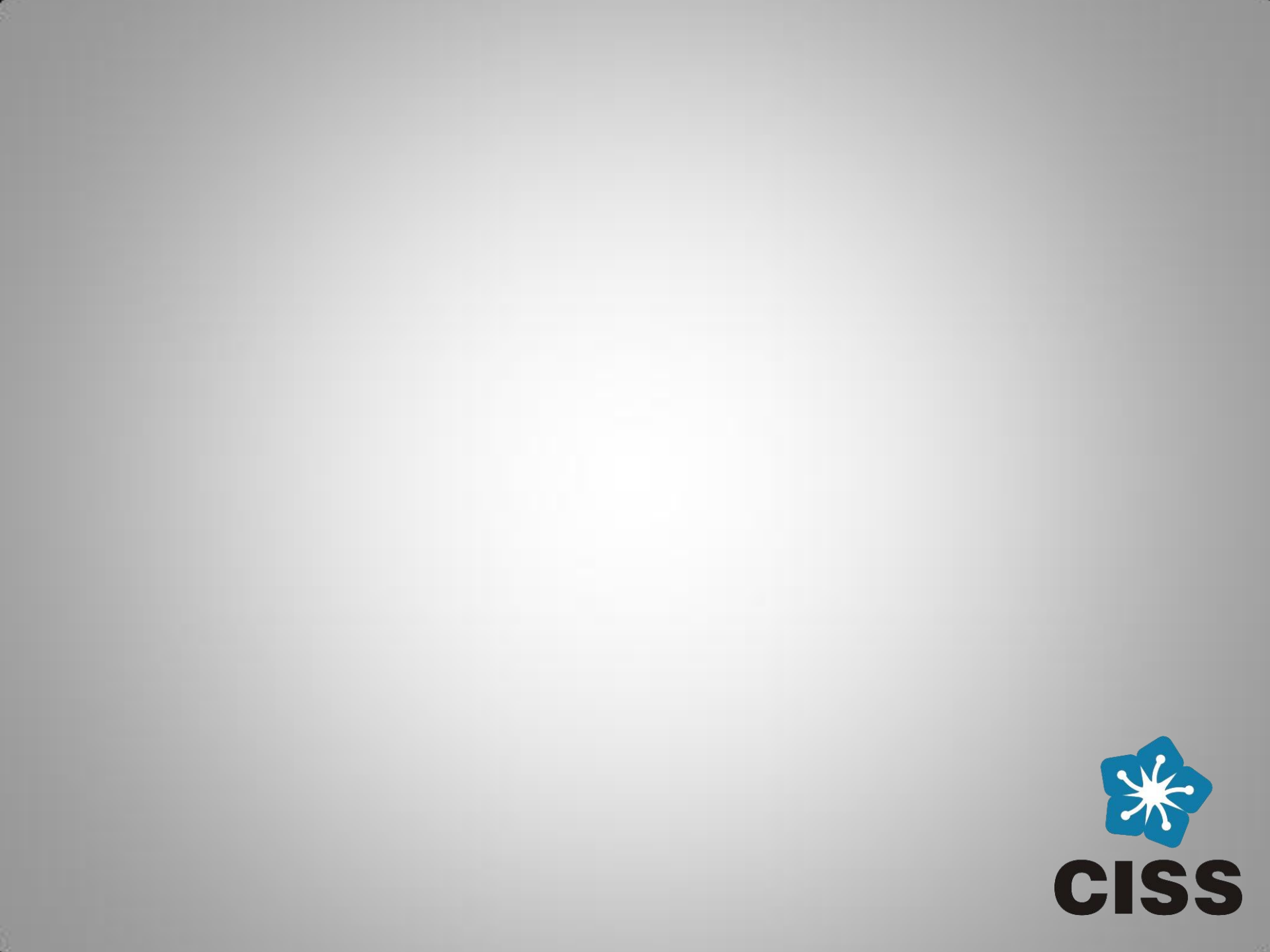

CISS

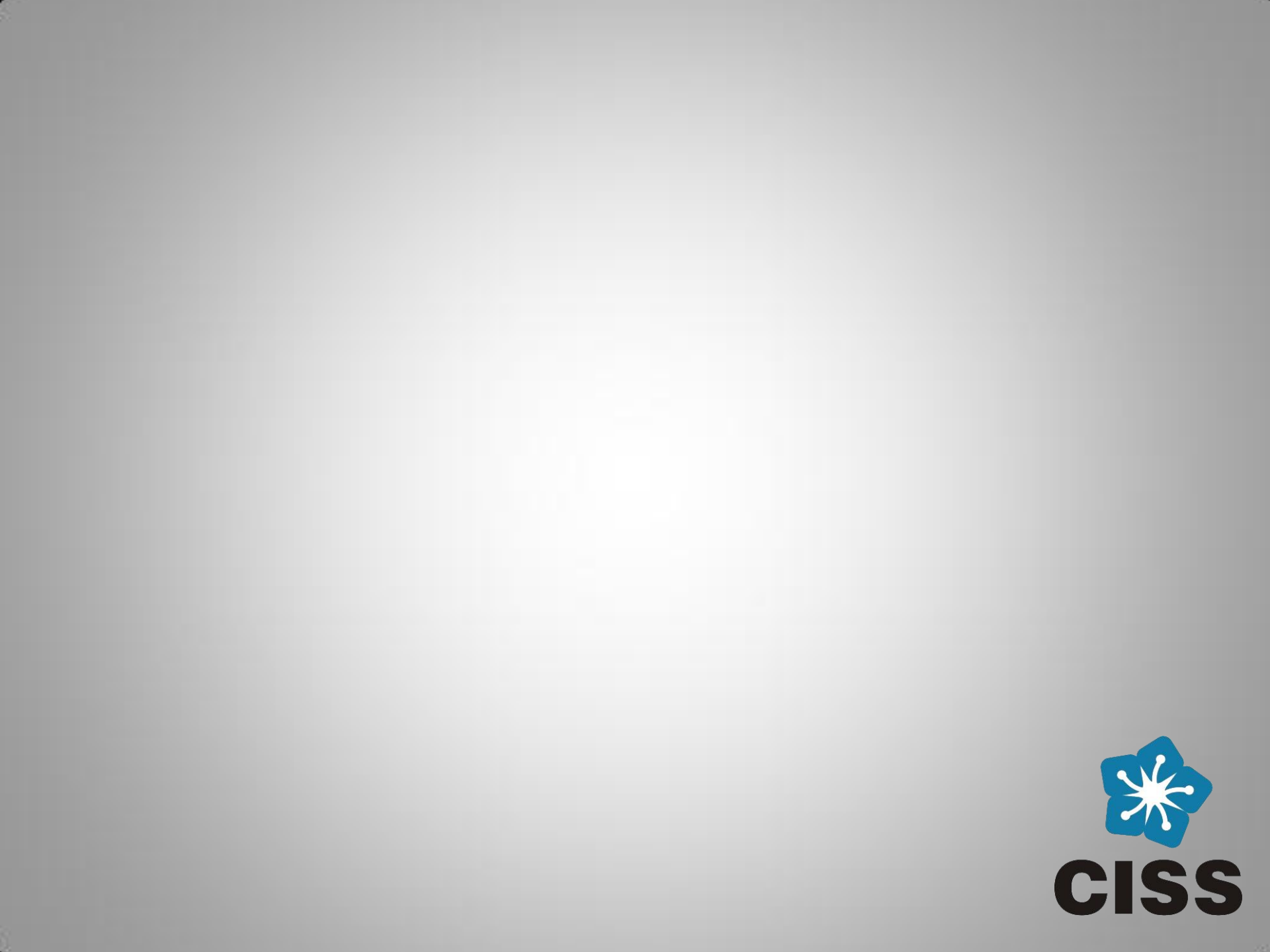

CISS

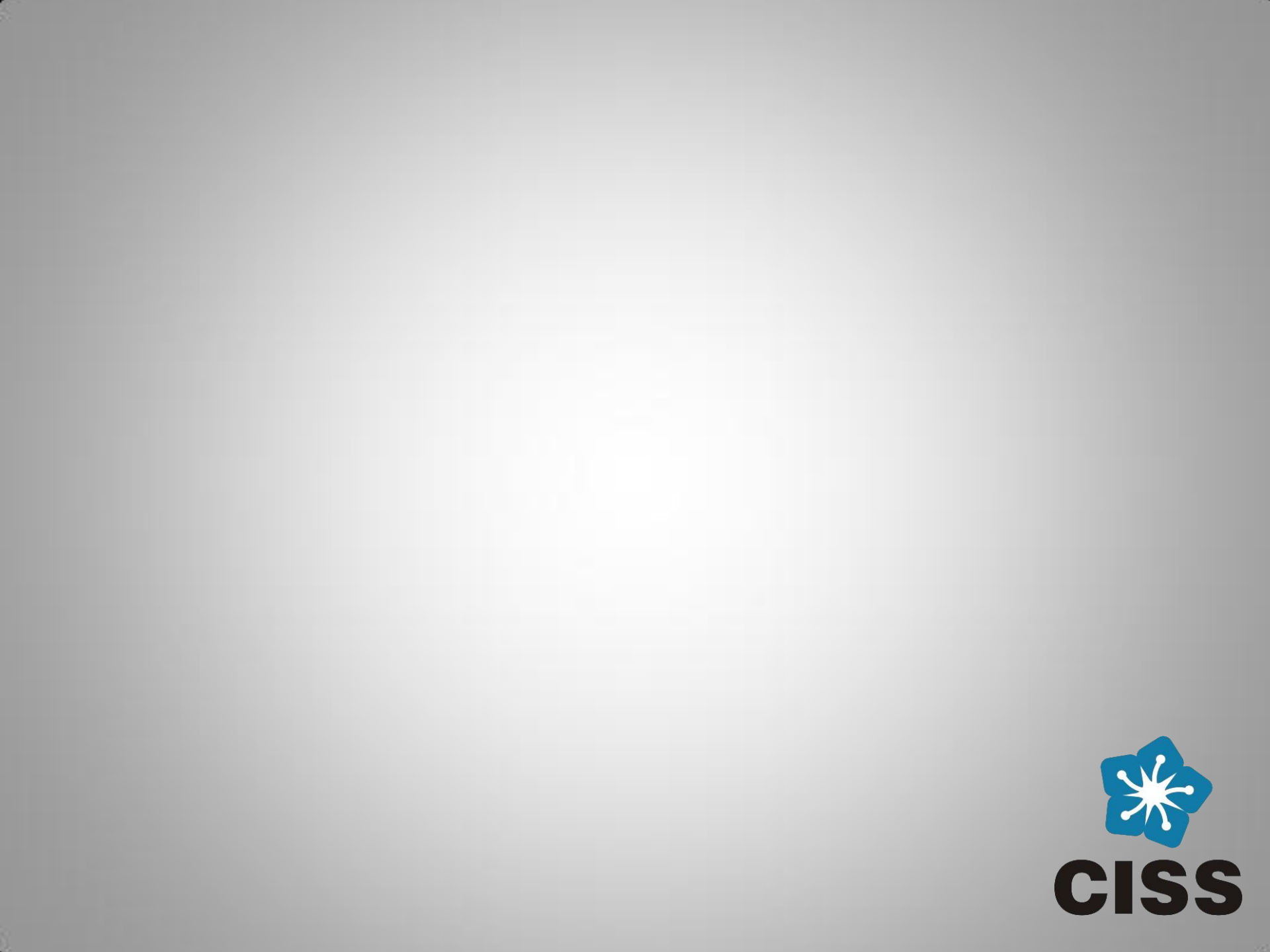

CISS

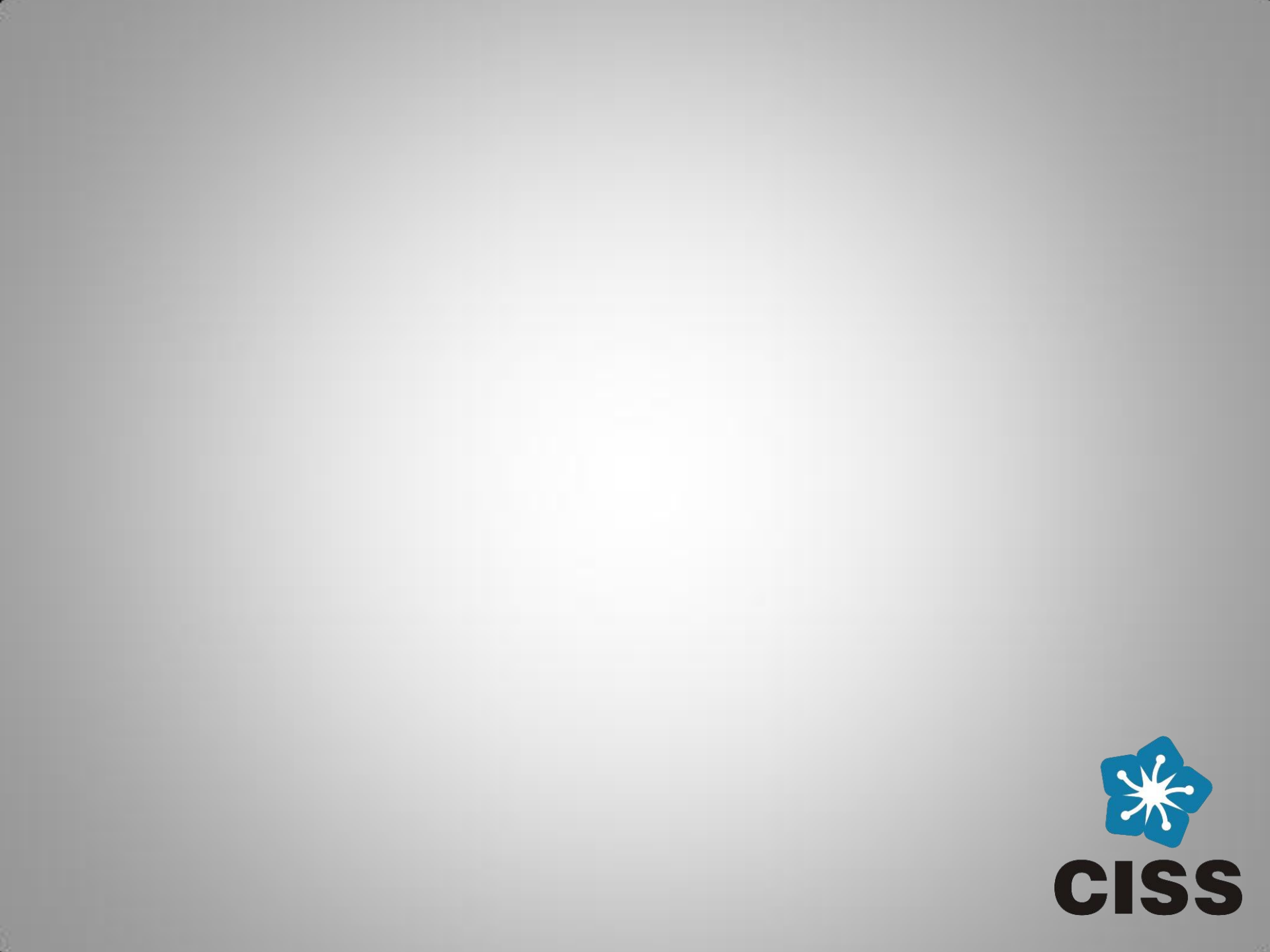

CISS

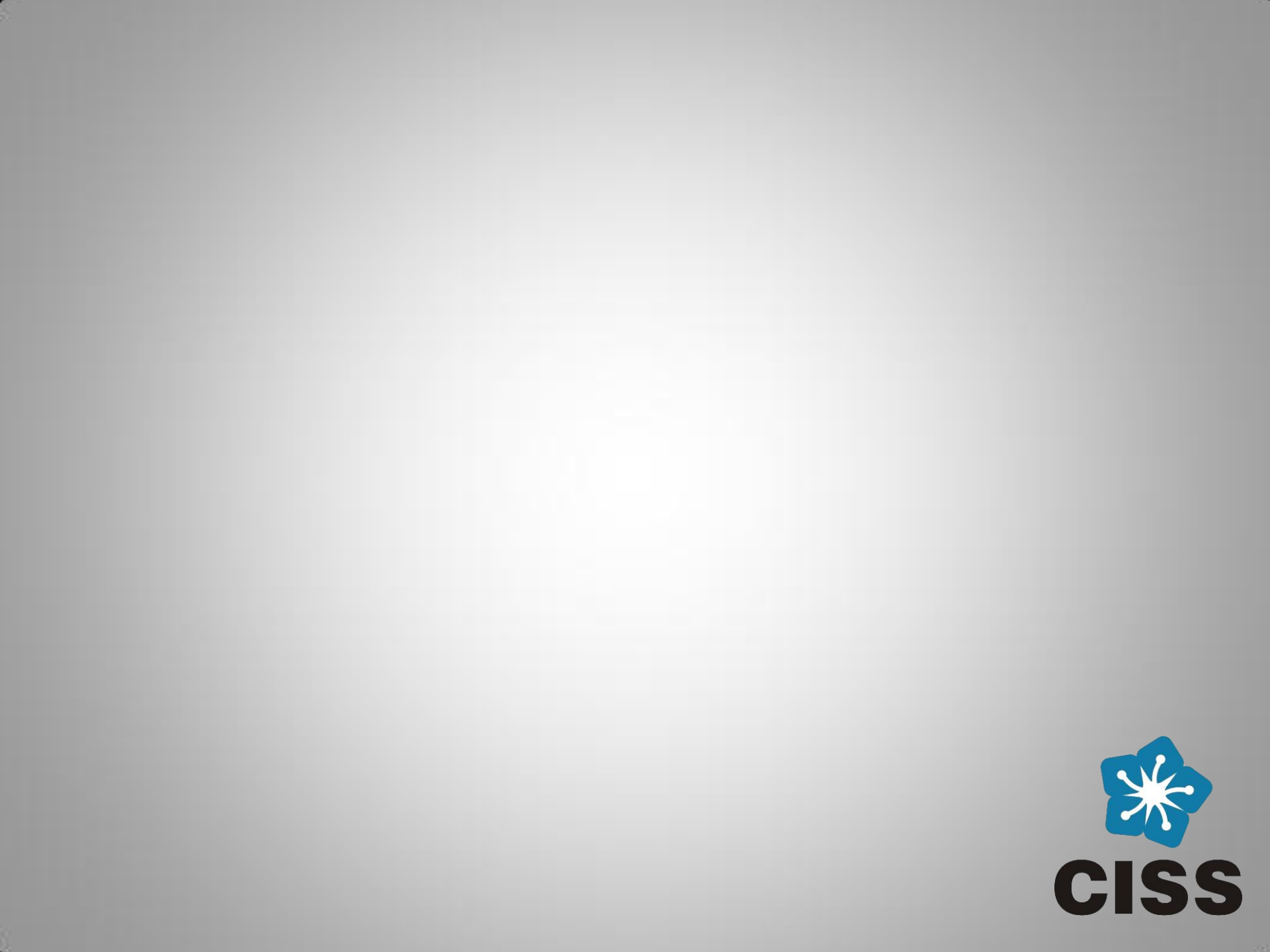

CISS

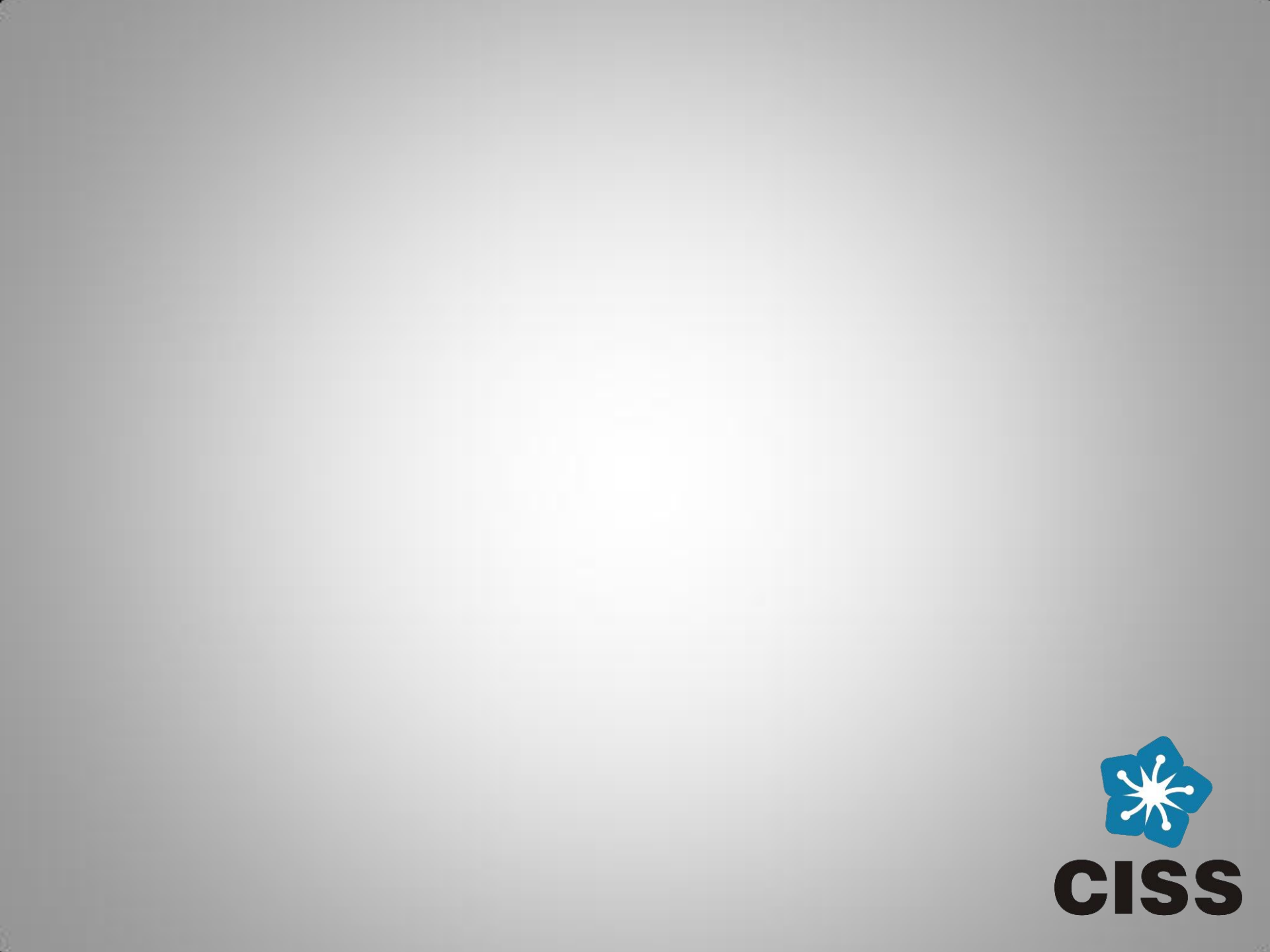

CISS

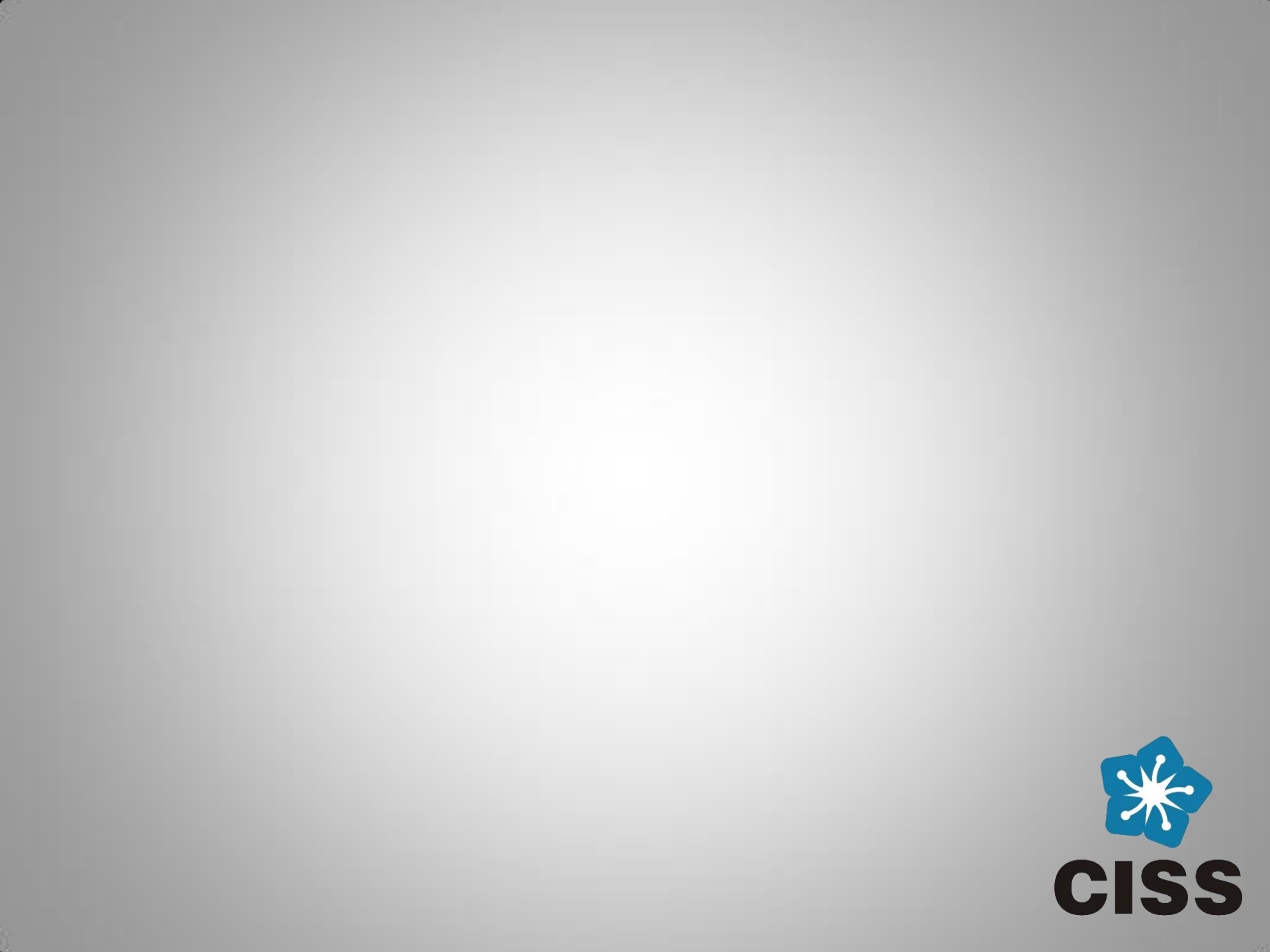

CISS


CISS


CISS


CISS


CISS


CISS